UNIVERSITÄT PADERBORN
Die Universität der Informationsgesellschaft
Fakultät für Elektrotechnik, Informatik und Mathematik

Diplomarbeit

Untersuchung von Clustering-Algorithmen für die Kullback-Leibler-Divergenz

Daniel Kuntze
E-Mail: kuntze@uni-paderborn.de

Paderborn, den 12. Juli 2007

vorgelegt bei
Prof. Dr.rer.nat. Johannes Blömer

Meinen Eltern

Inhaltsverzeichnis

1. Einleitung **1**

2. Grundlagen **5**
2.1. Clusteranalyse 5
2.2. Abstandsmaße 7
2.2.1. Quadrierte L_2-Norm 8
2.2.2. Spezielle Mahalanobis-Distanz 9
2.2.3. Kullback-Leibler-Divergenz 10

3. Algorithmen **15**
3.1. Partitionierende Clustering-Algorithmen 15
3.1.1. k-means 16
3.2. Hierarchische Clustering-Algorithmen 16
3.2.1. Agglomeratives Clustering 17
3.2.2. Divisive-Clustering 19

4. Analyse **21**
4.1. Untersuchungen zur quadrierten L_2-Norm 21
4.2. Clustering-Algorithmen im Vergleich 23
4.2.1. Markov-Ketten 23
4.2.2. Clusterzentren 25
4.2.3. Kostenfunktion 25
4.2.4. Abstandsfunktion 25
4.2.5. Eingabedateien 26
4.2.6. Ergebnis 26
4.3. Bad-Case-Szenario (agglomerativ) 28
4.3.1. Diagonale Mahalanobis-Distanz 30
4.3.2. Kullback-Leibler-Divergenz 34
4.4. Bad-Case-Szenario (Divisive) 37

5. Zusammenfassung **43**

A. Anhang **45**
A.1. Implementierung 45
A.1.1. Basisklassen 45
A.1.2. Bereits unterstützte Datentypen 46
A.1.3. Implementierte Algorithmen 48

Abbildungsverzeichnis

3.1. Hierarchisches Clustering einer 5-elementigen Punktmenge 17

4.1. Laufzeiten untersuchter Clustering-Algorithmen 27
4.2. Erwarteter Kompressionsverlust für die Textdatei 28
4.3. Erwarteter Kompressionsverlust für die Bilddatei 29
4.4. Erwarteter Kompressionsverlust für die Programmdatei 29
4.5. Die Punktmengenkonstruktion im Fall $n = 2$ 31
4.6. Die Punktmenge $W_0 \cup W_1$ im Fall $n = 2$ 38

A.1. Klassendiagramm für die CluE-Implementierung 47

Symbolverzeichnis

$\emptyset$	Leere Menge
$\mathbb{N}$	Menge der natürlichen Zahlen $\{1, 2, 3, \ldots\}$
$\mathbb{R}$	Menge der reellen Zahlen
$[a, b]$	Abgeschlossenes Interval mit den Grenzen $a < b$
(a, b)	Offenes Interval mit den Grenzen $a < b$
$n!$	Fakultät von n
$\binom{n}{k}$	Binomialkoeffizient n über k
$\log_b a$	Logarithmus von a zur Basis b
$\ln a$	natürlicher Logarithmus von a (zur Basis e $\approx 2,71828\ldots$)
L_2^2-Norm	Quadrierte L_2-Norm $\|\cdot\|^2$
$\mathbb{R}_+^n$	n-dimensionaler Vektorraum über der Menge der reellen Zahlen größer 0
x^T	Transponierter Vektor von x
$\langle x, y \rangle$	Skalarprodukt der Vektoren x und y
$\|M\|$	Kardinalität der Menge M
$M \times N$	Karthesisches Produkt der Mengen M und N
M^n	n-faches karthesisches Produkt der Menge M
$M \cap N$	Schnittmenge von M mit N
$M \cup N$	Vereinigungsmenge von M und N
$M \uplus N$	Multimengen-Vereinigung von M und N
$M \setminus N$	Differenzmenge M ohne N
$\mathcal{P}$	Potenzmenge der Menge M
$\mathcal{O}(f)$	Menge der Funktionen, die asymptotisch gleich stark oder schwächer wachsen als f
$P(X = a)$	Wahrscheinlichkeit, dass die Zufallsvariable X den Wert a annimmt

1. Einleitung

In der griechischen Mythologie bezeichnet der Begriff Kosmos[1] die (Welt-)Ordnung und bildet das Gegenstück zum Chaos[2], einem Zustand vollständiger Unordnung oder Verwirrung. Verwenden wir also Kosmos als Synonym für das Universum oder die Welt, schreiben wir ihr gleichzeitig eine innere Ordnung zu. Ein Ansatz, diese Ordnung zu erfassen, könnte sein, die Welt sinnvoll zu unterteilen. Die einfachste Einteilung wäre vielleicht die in ‚Ich' und ‚der Rest der Welt'. Man kann sich natürlich komplexere Kategorien vorstellen, aber unabhängig davon liegt die Sinnhaftigkeit dabei immer im Auge des Betrachters. So steht in einer gewissen chinesischen Enzyklopädie mit dem Titel „Himmlischer Warenschatz wohltätiger Erkenntnisse" geschrieben, dass sich die Tiere wie folgt gruppieren:

a) Tiere, die dem Kaiser gehören
b) einbalsamierte Tiere
c) gezähmte
d) Milchschweine
e) Sirenen
f) Fabeltiere
g) herrenlose Hunde
h) in diese Gruppierung gehörige
i) die sich wie Tolle gebärden
j) unzählbare
k) die mit einem ganz feinen Pinsel aus Kamelhaar gezeichnet sind
l) und so weiter
m) die den Wasserkrug zerbrochen haben
n) die von weitem wie Fliegen aussehen

Der Philosoph Michel Foucault zieht diese Einteilung in seiner Abhandlung „Die Ordnung der Dinge" [Foucault 2002] als Beispiel für die Fragwürdigkeit jeglicher Kategoriensysteme heran.

[1] griech. *κόσμος*
[2] griech. *χάος*

Ein Beispiel für eine ambitioniertere Einteilung der Tierwelt ist die Taxonomie in der Biologie, in der Lebewesen anhand ihrer Verwandschaftsbeziehungen hierarchisch geordnet werden. Dort hat die sogenannte Clusteranalyse (cluster (engl.) = Anhäufung), mit der wir uns in dieser Arbeit näher beschäftigen wollen, ihren historischen Ursprung.

Die Clusteranalyse ist ein Verfahren zur Ermittlung von Gruppen (Clustern) von Objekten, deren Eigenschaften oder ihre Ausprägungen bestimmte Ähnlichkeiten oder Unähnlichkeiten aufweisen. Die Informatik liefert hierfür eine Vielzahl von automatischen Berechnungsverfahren, von denen auf einige in Kapitel 3 näher eingegangen wird. Eine häufige Anwendung für die dort beschriebenen Clustering-Algorithmen ist die Datenreduzierung. Dabei geht es darum, eine große Menge von Daten zu reduzieren, bevor sie verarbeitet wird, um so Speicherplatz und/oder Rechenzeit einzusparen. Allerdings nimmt man dafür in der Regel in Kauf, dass das Ergebnis verfälscht wird. Ein Clustering-Algorithmus gruppiert dazu die ursprüngliche Datenmenge zu Clustern und anschließend wird für jeden Cluster ein Stellvertreter berechnet oder ausgewählt. Auf der Menge der Stellvertreter werden dann die eigentlichen Berechnungen ausgeführt.

Um diese Vorgehensweise zu veranschaulichen stelle man sich einen kleinen Jungen in einem großen, bunt gemischten Obstgarten vor. Der Junge möchte herausfinden, welches Obst ihm dort wohl am besten schmecken würde. Er könnte nun von einem Baum zum nächsten gehen, eine Frucht pflücken und probieren, bis er von jedem Baum probiert hat. Wenn der Obstgarten nur groß genug ist, wird er davon aber Bauchweh bekommen, bevor er auch nur annähernd von jedem Baum probiert hat. Würde er sich aber vorher die Bäume genauer ansehen und nach dem Aussehen der Früchte unterscheiden, stellte sich vielleicht heraus, dass es im ganzen Garten nur Apfel-, Birn- und Kirschbäume gibt. Dann wäre es schon damit getan, drei Früchte zu probieren.

Eine Verfälschung des Ergebnisses könnte sich in diesem Fall z. B. dadurch ergeben, dass der Junge einen Apfelbaum mit einer sauren Apfelsorte erwischt und nie von einem Baum mit süßen Äpfeln probiert.

In der Informatik gibt es sehr vielfältige Anwendungsgebiete für Clustering-Algorithmen: Bildverarbeitung, Sensornetzwerke oder Datenkompression, um nur einige zu nennen. Ein einfaches Beispiel ist die Partitionierung einer Punktmenge in eine gegebene Anzahl Cluster mit möglichst geringem Durchmesser. Wesentlich schwieriger wäre es dagegen, eine Menge von Porträtfotos nach Ähnlichkeit zu clustern, wobei in diesem Fall zunächst einmal genau zu definieren wäre, was mit Ähnlichkeit gemeint ist.

Eine spezielle Anwendung von Clustering-Algorithmen, mit der wir uns in dieser Arbeit näher beschäftigen wollen, ist das Clustern von Wahrscheinlichkeitsverteilungen. In der Codierungstheorie tritt vereinfacht ausgedrückt das folgende Problem auf. Je genauer man bei der Kompression einer Quelle (z. B. einer Datei) voraussagen kann, welches Symbol als nächstes gelesen wird, desto besser ist der Kompressionsgrad. Allerdings bezahlt man ein genaueres statistisches Modell mit einer größeren Anzahl an Wahrscheinlichkeitsverteilungen, die zusätzlich gespeichert

werden müssen. Daher wäre es wünschenswert, die Anzahl der Wahrscheinlichkeitsverteilungen zu reduzieren, ohne dabei einen allzu großen Kompressionsverlust hinnehmen zu müssen.

Dies sind ideale Voraussetzungen für den Einsatz von Clustering-Algorithmen. Allerdings müssen wir die Ähnlichkeit von Wahrscheinlichkeitsverteilungen in der Art angeben können, dass ähnliche Verteilungen einen geringen Kompressionsverlust erzeugen, wenn sie durch einen Stellvertreter ersetzt werden. Diese Ähnlichkeit werden wir durch die sogenannte Kullback-Leibler-Divergenz ausdrücken.

Eine Einführung in die Grundlagen der Clusteranalyse wird in Kapitel 2 gegeben. Dort wird auch die Kullback-Leibler-Divergenz definiert und ein Zusammenhang zur euklidischen Norm hergestellt. In Kapitel 3 werden dann drei Clustering-Verfahren vorgestellt, von denen zwei in Kapitel 4 analysiert werden. Kernstück der Analyse ist die Konstruktion von Bad-Case-Szenarien, in denen die Clustering-Algorithmen beweisbar schlechte Ergebnisse liefern.

2. Grundlagen

2.1. Clusteranalyse

Wir beginnen mit einer Beschreibung des in der Clusteranalyse zu lösenden Problems, wie es im Rahmen dieser Arbeit diskutiert werden soll. Als Eingabe ist eine beliebige endliche Menge von Objekten gegeben, die auf eine endliche Anzahl k von Teilmengen (Clustern) verteilt werden soll. Dies soll so geschehen, dass eine gegebene Kostenfunktion minimiert wird. Außerdem lassen wir nicht zu, dass ein Cluster kein Objekt enthält. Das Ergebnis nennt man dann ein k-Clustering. Wir fassen dies zunächst in einer allgemeinen Problemdefinition zusammen.

Problem 2.1 (k-Clustering-Problem, allgemein)
Sei P eine nicht leere Menge und $X \subseteq P$ eine endliche Teilmenge. Außerdem sei $k \in \mathbb{N}$ und eine Kostenfunktion

$$\text{cost} : \{1, \ldots, k\}^X \to \mathbb{R}$$

auf der Menge der Funktionen von X nach $\{1, \ldots, k\}$ gegeben.

Gesucht ist dann eine surjektive Abbildung

$$c : X \to \{1, \ldots, k\},$$

die cost minimiert.

Mit P bezeichnen wir den Probenraum, dem unsere zu clusternden Eingabeobjekte X entstammen. Die surjektive Abbildung c beschreibt das k-Clustering, indem sie jedem Element aus X einen Index von 1 bis k zuordnet.

Als nächstes wollen wir einige Spezialisierungen des allgemeinen k-Clustering-Problems erarbeiten. In der Problemdefinition 2.1 werden keine näheren Angaben zu der Kostenfunktion gemacht. Das macht es schwierig, wenn nicht unmöglich, einen nicht-trivialen[1] Algorithmus zu entwerfen, der die Kostenfunktion minimiert. Wir werden daher die Kostenfunktion einschränken. Genauer gesagt geben wir zwei spezielle Kostenfunktionen an, die auf einem sogenannten Abstandsmaß basieren. Dazu führen wir zunächst den Begriff des Abstandsmaßes ein.

Definition 2.2 *Sei eine beliebige Menge X gegeben. Ein Abstandsmaß auf X ist*

[1] Trivial wäre z. B. eine vollständige Suche auf der gesamten Eingabe oder auf einer zufälligen Stichprobe.

eine Abbildung

$$d : X^2 \to \mathbb{R}.$$

Für $x, y \in X$ heißt $d(x, y)$ der Abstand von x zu y. Dabei ist zu beachten, dass der Abstand im Allgemeinen gerichtet ist, d. h. die Abbildung d nicht symmetrisch sein muss. Für eine Teilmenge $U \subseteq X$ bezeichnen wir

$$d(U, y) := \sum_{x \in U} d(x, y)$$

als den Abstand von U zu y.

Unsere erste Kostenfunktion berechnet den maximalen Clusterdurchmesser des berechneten k-Clusterings bzgl. eines gegebenen Abstandsmaßes. Wir erhalten daraus folgende Problemdefinition.

Problem 2.3 (k-Clustering-Problem, Durchmesser minimierend)
Sei P eine nicht leere Menge mit einem darauf definierten Abstandsmaß d. Außerdem sei $X \subseteq P$ eine endliche Teilmenge von P und $k \in \mathbb{N}$. Gesucht ist dann eine surjektive Abbildung

$$c : X \to \{1, \ldots, k\},$$

die die Kostenfunktion

$$\text{cost}_{\varnothing}(c) = \max_{\substack{x,y \in X \\ c(x)=c(y)}} d(x, y)$$

minimiert.

Für die zweite Kostenfunktion benötigen wir sogenannte Clusterzentren. Diese sind Stellvertreter, die die Elemente eines Clusters in sinnvoller Weise repräsentieren. Für eine Punktmenge könnte dies z. B. der Schwerpunkt sein. Da die Clusterzentren auch unabhängig von der verwendeten Kostenfunktion von Interesse sein können, geben wir nun eine entsprechende Problemdefinition an. Anschließend widmen wir uns dann unserer zweiten Kostenfunktion.

Problem 2.4 (k-Clustering-Problem, mit Zentren)
Ist zur Lösung des allgemeinen k-Clustering-Problems (Problem 2.1) zusätzlich eine Abbildung

$$z : \mathcal{P}(P) \setminus \{\emptyset\} \longrightarrow P$$

gegeben, dann sprechen wir vom k-Clustering-Problem mit Zentren.

Die Abbildung z ordnet jeder nicht leeren Teilmenge des Probenraums P ein Zentrum zu.

Die zweite Kostenfunktion minimiert die Summe der Abstände aller Elemente zu ihren jeweiligen Clusterzentren. Es handelt sich hier also um eine Spezialisierung des k-Clustering-Problems mit Zentren (Problem 2.4). Wir geben dennoch eine komplette formale Definition an.

Problem 2.5 (k-Clustering-Problem, mit Zentrumskosten)
Sei P eine nicht leere Menge mit einem darauf definierten Abstandsmaß d und einer Abbildung

$$z : \mathcal{P}(P) \setminus \{\emptyset\} \longrightarrow P.$$

Außerdem sei $X \subseteq P$ eine endliche Teilmenge von P und $k \in \mathbb{N}$ gegeben. Gesucht ist dann eine surjektive Abbildung

$$c : X \to \{1, \ldots, k\},$$

die die Kostenfunktion

$$\text{cost}_{\text{ZK}}(c) = \sum_{x \in X} d(x, z(C_{c(x)})) = \sum_{i=1}^{k} \sum_{\substack{x \in X \\ c(x)=i}} d(x, z(C_i))$$

minimiert, wobei

$$C_i = \{x \in X | c(x) = i\}$$

für $i = 1, \ldots, k$ gilt.

C_i bezeichnet hier den i-ten Cluster.

Es ließen sich natürlich noch beliebig viele weitere Kostenfunktionen und zugehörige Problemdefinitionen angeben. Meistens basiert die Kostenfunktion dabei auf einem Abstandsmaß und es wird versucht, einen bestimmten Abstand zu minimieren. Es kommt dann darauf an, ein zum Einsatzgebiet passendes Abstandsmaß zu definieren. Um das Beispiel aus der Einleitung aufzugreifen, müssten wir beim Clustern der Porträtfotos dafür sorgen, dass ähnliche Fotos einen geringen Abstand zueinander haben.

2.2. Abstandsmaße

In diesem Abschnitt werfen wir einen Blick auf drei Abstandsmaße, mit denen wir uns in dieser Arbeit näher beschäftigen wollen. Auf dem $\mathbb{R}^n$ betrachten wir die quadrierte L_2-Norm (L_2^2-Norm) und einen Spezialfall der Mahalanobis-Distanzen. Beim dritten Abstandsmaß handelt es sich um die Kullback-Leibler-Divergenz auf dem Wahrscheinlichkeits-Simplex

$$\Delta_n := \left\{ p = (p_1, \ldots, p_n) \in \mathbb{R}^n \;\middle|\; p_i \geq 0 \text{ für alle } i \text{ und } \sum_{i=1}^{n} p_i = 1 \right\}$$

und in einer verallgemeinerten Variante auf der Menge $\mathbb{R}_+^n$.

2.2.1. Quadrierte L_2-Norm

Wir wählen als Probenraum den n-dimensionalen reellen Vektorraum $\mathbb{R}^n$ mit $n \in \mathbb{N}$. Für den $\mathbb{R}^n$ definieren wir den Schwerpunkt einer Punktmenge.

Definition 2.6 (Schwerpunkt) *Sei ein endliches $C \subseteq \mathbb{R}^n$ gegeben. Dann heißt*

$$\operatorname{cog}(C) := \frac{1}{|C|} \sum_{x \in C} x$$

der Schwerpunkt (engl. center of gravity) von C.

Als Abstandsmaß d wählen wir in diesem Abschnitt die L_2^2-Norm. Damit ist der Abstand $d(x, y)$ zweier Punkte $x, y \in \mathbb{R}^n$ definiert als das Quadrat der L_2-Norm des Differenzvektors $x - y$, also:

$$d(x, y) \;=\; \|x - y\|^2 \;=\; \langle x - y, x - y \rangle \;=\; \langle x, x \rangle - \langle y, y \rangle - \langle x - y, 2y \rangle \tag{2.1}$$

Das folgende Lemma stellt einen Zusammenhang zwischen den Kosten eines Clusters und seinem Schwerpunkt her.

Lemma 2.7 *Für einen beliebigen Cluster $C \subset \mathbb{R}^n$ mit Clusterzentrum $z \in \mathbb{R}^n$ und der L_2^2-Norm als Abstandsmaß d gilt*

$$d(C, z) \;=\; d(C, \operatorname{cog}(C)) \;+\; |C| \cdot d(\operatorname{cog}(C), z).$$

Korrolar 2.8 *Sei die Kostenfunktion $\operatorname{cost}_{\mathrm{ZK}}$ aus Problem 2.5 gegeben. Dann bildet der Schwerpunkt $\operatorname{cog}(C)$ eines Clusters $C \subset \mathbb{R}^n$ das optimale Clusterzentrum bezüglich der L_2^2-Norm als Abstandsmaß.*

Beweis (Lemma 2.7) Der Beweis erfolgt durch Nachrechnen:

$$\begin{aligned}
& d(C, z) \;-\; d(C, \operatorname{cog}(C)) \\
= \;& \sum_{x \in C} \|x - z\|^2 - \sum_{x \in C} \|x - \operatorname{cog}(C)\|^2 \\
\overset{(2.1)}{=} \;& \sum_{x \in C} (\langle x, x \rangle - \langle z, z \rangle - \langle x - z, 2z \rangle) \\
& - \sum_{x \in C} (\langle x, x \rangle - \langle \operatorname{cog}(C), \operatorname{cog}(C) \rangle - \langle x - \operatorname{cog}(C), 2 \operatorname{cog}(C) \rangle) \\
= \;& \sum_{x \in C} (\langle \operatorname{cog}(C), \operatorname{cog}(C) \rangle - \langle z, z \rangle - \langle x - z, 2z \rangle + \langle x - \operatorname{cog}(C), 2 \operatorname{cog}(C) \rangle)
\end{aligned}$$

$$
\begin{aligned}
&= |C|(\langle \operatorname{cog}(C), \operatorname{cog}(C)\rangle - \langle z, z\rangle) - \left\langle \sum_{x\in C}(x-z), 2z \right\rangle \\
&\quad + \left\langle \underbrace{\sum_{x\in C}(x-\operatorname{cog}(C))}_{=0}, 2\operatorname{cog}(C) \right\rangle \\
&= |C| \left(\langle \operatorname{cog}(C), \operatorname{cog}(C)\rangle - \langle z, z\rangle - \left\langle \frac{1}{|C|}\sum_{x\in C}(x-z), 2z \right\rangle \right) \\
&= |C|(\langle \operatorname{cog}(C), \operatorname{cog}(C)\rangle - \langle z, z\rangle - \langle \operatorname{cog}(C) - z, 2z\rangle) \\
&= |C| \, \| \operatorname{cog}(C) - z\|^2
\end{aligned}
$$

□

2.2.2. Spezielle Mahalanobis-Distanz

Wir definieren nun kurz die Mahalanobis-Distanzen und schränken uns dann auf einen Spezialfall ein. Die L_2^2-Norm wird sich dabei als ein Spezialfall dieses Spezialfalls herausstellen.

Definition 2.9 (Mahalanobis-Distanz) *Sei $n \in \mathbb{N}$ und $A \in \mathbb{R}^{n\times n}$ eine positiv definite Matrix. Für $x, y \in \mathbb{R}^n$ heißt dann*

$$d(x,y) := (x-y)^T \cdot A \cdot (x-y)$$

die Mahalanobis-Distanz von x zu y, wenn A das Inverse der Kovarianzmatrix ist.

Die speziellen Matrizen, mit denen wir uns in diesem Abschnitt beschäftigen wollen, erfüllen das Kriterium, dass sie das Inverse einer Kovarianzmatrix sind. Was mit der Kovarianzmatrix gemeint ist, ist für unsere Untersuchungen aber nicht von Bedeutung. Obige Definition dient lediglich dazu, das als nächstes definierte Abstandsmaß als Spezialfall einer Mahalanobis-Distanz zu erkennen. Mehr zu Mahalanobis-Distanzen kann bei [Mahalanobis 1936] nachgelesen werden.

Definition 2.10 (Diagonale Mahalanobis-Distanz) *Sei $D \in \mathbb{R}^{n\times n}$ für $n \in \mathbb{N}$ gegeben durch*

$$D := \begin{pmatrix} c_1 & & 0 \\ & \ddots & \\ 0 & & c_n \end{pmatrix}$$

mit $c_i > 0$ für $i = 1, \ldots, n$.

Basierend auf D definieren wir nun

$$d(x,y) := (x-y)^T \cdot D \cdot (x-y) = \sum_{i=1}^{n} c_i (x_i - y_i)^2$$

für $x, y \in \mathbb{R}^n$. Den Abstand $d(x, y)$ nennen wir diagonale Mahalanobis-Distanz von x zu y.

Bemerkung 2.11
Wenn wir die Einheitsmatrix für die Matrix D aus Definition 2.10 einsetzen, erhalten wir mit

$$d(x, y) = (x - y)^T \cdot (x - y) = \sum_{i=1}^{n} (x_i - y_i)^2 = \|x - y\|^2$$

die L_2^2-Norm als Spezialfall einer diagonalen Mahalanobis-Distanz.

2.2.3. Kullback-Leibler-Divergenz

In diesem Abschnitt soll es um ein Abstandsmaß für Wahrscheinlichkeitsverteilungen und dessen Verallgemeinerung auf der Menge $\mathbb{R}_+^n$ gehen. Es handelt sich dabei um die Kullback-Leibler-Divergenz, die wir kurz motivieren wollen. Für eine tiefergehende Auseinandersetzung mit dem Thema sei auf [Kullback 1959] und [Kullback und Leibler 1951] verwiesen.

Wir beginnen mit einem Ausflug zu den Anfängen der Informationstheorie, die im Jahr 1948 von Claude E. Shannon durch seine Arbeit „A Mathematical Theory of Communication“ [Shannon 1948] begründet wurde. Darin definiert Shannon die Entropie einer diskreten gedächtnislosen Quelle.

Definition 2.12 *Sei X eine Zufallsvariable über einem höchstens abzählbaren Alphabet $A = \{a_1, a_2, \ldots\}$ mit $p_i = p(a_i) = P(X = a_i)$ und $\sum_{i=1}^{|A|} p_i = 1$. Dann ist die Entropie $H(X)$ von X gegeben durch*

$$H(X) := -\sum_{i=1}^{|A|} p_i \log_2 p_i.$$

Die Entropie beschreibt den Informationsgehalt einer Quelle. Diese Interpretation sieht man leichter ein, wenn man die folgende alternative Darstellung der Entropie betrachtet:

$$H(X) = \sum_{i=1}^{|A|} p_i \cdot I_2(p_i),$$

wobei mit $I_2(p_i) := -\log_2 p_i$ die Selbstinformation von a_i in Bits bezeichnet wird. Das bedeutet, ein optimaler binärer Code müsste das Symbol a_i mit $I_2(p_i)$ Bits codieren und würde im Erwartungswert $H(X)$ viele Bits benötigen um ein Symbol von X zu codieren. Wir ignorieren hier die Tatsache, dass weder $I_2(p_i)$ noch $H(X)$ eine natürliche Zahl sein müssen. Eine genauere Auseinandersetzung mit den Grundlagen der Codierungstheorie kann beispielsweise bei [Sayood 2005] nachgelesen werden.

Wenn wir den Logarithmus zu einer anderen Basis betrachten, erhalten wir den Informationsgehalt lediglich in einer anderen Einheit. Beispielsweise misst

$$I_{\mathrm{e}}(p_i) := -\ln p_i$$

die Selbstinformation in Nats. Um einige Berechnungen zu vereinfachen, werden wir von nun an nur noch den natürlichen Logarithmus betrachten und den Informationsgehalt in Nats messen. Die Ergebnisse unterscheiden sich dabei wegen

$$\log_2 = \frac{\ln}{\ln 2}$$

nur um einen konstanten Faktor.

Wir definieren nun mit Y eine zweite Zufallsvariable über A mit Wahrscheinlichkeiten $q_i = P(Y = a_i)$. Unser Interesse gilt den zusätzlich benötigten Nats eines optimalen Codes für Y, wenn wir mit ihm die Symbole der Zufallsvariablen X codieren. Der Erwartungswert für die benötigten Nats, den wir für diesen Fall mit $E(X|Y)$ bezeichnen, wäre dann

$$E(X|Y) \;=\; \sum_{i=1}^{|A|} p_i \cdot I_{\mathrm{e}}(q_i).$$

Für die zusätzlich benötigten Nats gilt dann:

$$\begin{aligned} E(X|Y) - H(X) = & -\sum_{i=1}^{|A|} p_i \,\ln q_i \;+\; \sum_{i=1}^{|A|} p_i \,\ln p_i \\ = & \sum_{i=1}^{|A|} p_i \,(\ln p_i \;-\; \ln q_i) \\ = & \sum_{i=1}^{|A|} p_i \,\ln \frac{p_i}{q_i} \end{aligned}$$

Diesen Wert nennt man die Kullback-Leibler Divergenz einer Wahrscheinlichkeitsverteilung $p = (p_1, p_2, \ldots)$ zu einer Verteilung $q = (q_1, q_2, \ldots)$.

Es folgt eine formale Definition für die Kullback-Leibler Divergenz auf dem Wahrscheinlichkeits-Simplex

$$\Delta_n \;=\; \left\{ p = (p_1, \ldots, p_n) \in \mathbb{R}^n \;\middle|\; p_i \geq 0 \text{ für alle } i \text{ und } \sum_{i=1}^{n} p_i = 1 \right\}.$$

Definition 2.13 *Seien $p, q \in \Delta_n$. Dann heißt*

$$D(p|q) := \sum_{i=1}^{n} p_i \ln \frac{p_i}{q_i}$$

die Kullback-Leibler Divergenz (KLD) der Verteilung p zur Verteilung q. Um die Wohldefiniertheit zu gewährleisten vereinbaren wir für $a, b \geq 0$

$$\ln \frac{a}{b} = \begin{cases} 0 & \text{falls } a = 0, \\ \infty & \text{falls } a > 0 \text{ und } b = 0. \end{cases}$$

Die zusätzliche Vereinbarung, die in Definition 2.13 getroffen wurde, ist auf der einen Seite nötig, damit die KLD auf dem Rand von Δ_n definiert ist. Auf der anderen Seite ist sie genau so, wie sie getroffen wurde, auch sinnvoll. Wir schauen uns dazu noch einmal die Motivation mit dem optimalen Code und den Zufallsvariablen X und Y an. Wir codieren wieder die Symbole von X mit einem optimalen Code für Y.

Wenn nun ein Symbol a_i nicht auftritt, weil $p_i = 0$ gilt, dann erzeugt der Code für Y auch keine Ausgabe. Dies gilt unabhängig davon, ob Y das Symbol erzeugen könnte, also unabhängig von q_i. Der entsprechende Summand wird aber durch unsere Vereinbarung in diesem Fall auch auf 0 gesetzt.

Tritt nun allerdings ein Symbol a_j auf, für das $q_j = 0$ gilt, dann kann es mit dem Code für Y nicht codiert werden. Dies drückt sich auch in der Tatsache aus, dass der Informationsgehalt eines Symbols gegen unendlich strebt, wenn seine Wahrscheinlichkeit gegen 0 geht:

$$I_{\mathrm{e}}(r) = -\ln r \xrightarrow{r \to 0} \infty$$

In diesem Fall wäre die KLD von p zu q durch unsere Vereinbarung auch unendlich groß. Dies ist eine sinnvolle Festlegung, die ausdrückt, dass sich p und q in diesem Fall alles andere als ähnlich sind.

Wir definieren nun die verallgemeinerte Kullback-Leibler-Divergenz als Abstandsmaß auf dem $\mathbb{R}^n_+$. Auf eine Einbeziehung des Randes verzichten wir bewusst, da wir ihn in unseren Untersuchungen nicht benötigen. Abgesehen davon geht die KLD aus Definition 2.13 als Spezialfall in der folgenden Definition auf. Daher wählen wir auch für beide Abstandsmaße die gleichen Bezeichner.

Definition 2.14 *Mit $D(\cdot|\cdot)$ bezeichnen wir die verallgemeinerte Kullback-Leibler-Divergenz (KLD) und definieren für $x, y \in \mathbb{R}^n_+$:*

$$D(x|y) := \sum_{i=1}^{n} \left(x_i \ln \frac{x_i}{y_i} - (x_i - y_i) \right)$$

Wir lesen $D(x|y)$ als KLD von x zu y.

Die Aussage des folgenden Lemmas entstammt sinngemäß [Ackermann u. a. 2007].

Lemma 2.15 *Seien $\beta, \gamma \in \mathbb{R}$ mit $0 < \beta < \gamma$ und $x, y \in [\beta, \gamma]^n \subset \mathbb{R}_+^n$ für ein $n \in \mathbb{N}$. Dann ist die KLD von x zu y wie folgt durch die L_2^2-Norm beschränkt:*

$$\frac{1}{2\gamma} \parallel x - y \parallel^2 \leq D(x|y) \leq \frac{1}{2\beta} \parallel x - y \parallel^2$$

Beweis Wir betrachten die zweimal differenzierbare, strikt konvexe Funktion

$$\varphi(t) = \sum_{i=0}^{n} t_i \ln t_i$$

und deren Taylor-Entwicklung um den Entwicklungspunkt y mit dem ersten Taylorpolynom $T_1(x)$ und dem ersten Restglied $R_1(x)$:

$$\varphi(x) = T_1(x) + R_1(x) = \varphi(y) + \sum_{i=1}^{n} \frac{\partial}{\partial t_i} \varphi(y)(x_i - y_i) + R_1(x) \tag{2.2}$$

Mit

$$\frac{\partial}{\partial t_i} \varphi(t) = \ln t_i + 1$$

können wir Gleichung (2.2) folgendermaßen nach $R_1(x)$ auflösen:

$$R_1(x) = \varphi(x) - T_1(x) = \varphi(x) - \varphi(y) - \sum_{i=1}^{n} (\ln y_i + 1)(x_i - y_i)$$

Durch Einsetzen der Funktionsdefinition von φ erhalten wir daraus:

$$\begin{aligned} R_1(x) &= \sum_{i=0}^{n} x_i \ln x_i - \sum_{i=0}^{n} y_i \ln y_i - \sum_{i=1}^{n} (x_i - y_i) \ln y_i - \sum_{i=1}^{n} (x_i - y_i) \\ &= \sum_{i=0}^{n} x_i \ln x_i - \sum_{i=0}^{n} x_i \ln y_i - \sum_{i=1}^{n} (x_i - y_i) \\ &= \sum_{i=0}^{n} \left(x_i \ln \frac{x_i}{y_i} - (x_i - y_i) \right) \end{aligned}$$

Das Restglied $R_1(x)$ der Taylorentwicklung um den Punkt y ist also gerade die KLD von x zu y (vergl. Definition 2.14). Es gilt also $R_1(x) = D(x|y)$.

Im folgenden sei

$$S_{xy} := \{ z \in \mathbb{R}_+^n \mid \exists \lambda \in [0, 1] \text{ mit } z = x + \lambda(y - x) \}$$

die Menge der Punkte auf der Verbindungsstrecke zwischen x und y. Wir betrachten $R_1(x)$ nun in der Restglieddarstellung von Lagrange. Diese besagt, dass ein $\xi \in S_{xy}$

existiert, so dass gilt:

$$\begin{aligned} R_1(x) &= \frac{1}{2}\sum_{i=1}^{n}(x_i - y_i)\frac{\partial}{\partial x_i}\sum_{j=1}^{n}(x_j - y_j)\frac{\partial}{\partial x_j}\varphi(\xi) \\ &= \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}(x_i - y_i)(x_j - y_j)\frac{\partial^2}{\partial x_i \partial x_j}\varphi(\xi) \end{aligned}$$

Für die partiellen Ableitungen von φ gilt aber

$$\frac{\partial^2}{\partial t_i \partial t_j}\varphi(t) = \begin{cases} \frac{\partial}{\partial t_i}(\ln t_j + 1) = 0 & \text{falls } i \neq j, \\ \frac{\partial}{\partial t_i}(\ln t_i + 1) = \frac{1}{t_i} & \text{falls } i = j. \end{cases}$$

für $i, j \in \{1, \ldots, n\}$. Damit erhalten wir:

$$D(x|y) = R_1(x) = \frac{1}{2}\sum_{i=1}^{n}(x_i - y_i)^2\frac{1}{\xi_i}$$

Wegen $x, y \in [\beta, \gamma]^n$ gilt auch $S_{xy} \subset [\beta, \gamma]^n$ und damit folgt auch $\xi \in [\beta, \gamma]^n$. Daraus folgen schließlich eine obere und eine untere Schranke für das Restglied und damit auch für die KLD:

$$\frac{1}{2\gamma}\sum_{i=1}^{n}(x_i - y_i)^2 \leq D(x|y) \leq \frac{1}{2\beta}\sum_{i=1}^{n}(x_i - y_i)^2$$

□

3. Algorithmen

Dieses Kapitel bietet eine Einführung in einige Clustering-Algorithmen, die anschließend im Kapitel 4 untersucht werden. Die Aufgabe eines Clustering-Algorithmus ist die Berechnung eines möglichst guten, d. h. wenig Kosten erzeugenden Clusterings. Dabei wird die optimale Lösung meistens nicht erreicht. Vielmehr handelt es sich bei Clustering-Algorithmen in der Regel um Approximationsalgorithmen, die versuchen, so nahe wie möglich an die optimale Lösung heranzukommen.

Bemerkung 3.1
In der Regel verwenden Clustering-Algorithmen eine Abstandsfunktion, um die gegebene Kostenfunktion zu minimieren. Diese Abstandsfunktion basiert im Normalfall auf dem gleichen Abstandsmaß wie die Kostenfunktion. Sie kann auch genau gleich dem verwendeten Abstandsmaß sein. Es ist aber wichtig, die beiden Begriffe zu unterscheiden, denn das Abstandsmaß fließt über die Kostenfunktion mit in die Problemdefinition ein, während die Abstandsfunktion lediglich ein Hilfsmittel des Algorithmus ist.

Wir beschränken uns in dieser Arbeit auf drei Algorithmen, einen partitionierenden Algorithmus und zwei hierarchische Algorithmen. Was das bedeutet, wird in den nächsten Abschnitten erklärt. Für einen breiteren Einstieg in die Materie sei z. B. auf [Mercer 2003] verwiesen.

3.1. Partitionierende Clustering-Algorithmen

Partitionierende Clustering-Algorithmen unterteilen eine Menge X von zu clusternden Objekten in eine gegebene Anzahl k von Partitionen (Clustern). Sie berechnen also eine Lösung des allgemeinen k-Clustering-Problems (Problem 2.1).

Um das globale Optimum für die Partitionierung zu ermitteln, müssten alle

$$S(|X|,k) = \frac{1}{k!}\sum_{i=1}^{k}(-1)^{(k-i)}\binom{k}{i}i^{|X|} \tag{3.1}$$

möglichen Partitionen einbezogen werden. $S(n,k)$ aus Gleichung (3.1) ist die Stirlingsche Zahl zweiter Art und beschreibt gerade die Anzahl möglicher Partitionen einer n-elementigen Menge in k nicht leere Teilmengen.

Ein einfacher Suchalgorithmus würde alle möglichen Partitionen nacheinander ausprobieren und damit schnell an die Grenze dessen stoßen, was man noch als effizient durchführbar bezeichnen könnte. Aber auch ein intelligentes Suchverfahren

gelangt für größere Mengen X schnell an die Grenzen des Machbaren. Daher berechnen partitionierende Clustering-Algorithmen in der Regel ein lokales Optimum. Es sind, wie bereits gesagt, Approximationsalgorithmen.

3.1.1. k-means

Der k-means-Algorithmus [MacQueen 1967] ist ein Standardverfahren, das sich immer noch großer Beliebtheit erfreut. Der Algorithmus löst das k-Clustering-Problem mit Zentren (Problem 2.4). Er benötigt eine Abstandsfunktion $f_d(\cdot, \cdot)$ (siehe Bemerkung 3.1) und eignet sich daher für Kostenfunktionen, die auf einem Abstandsmaß basieren.

Der Ablauf des k-means-Algorithmus auf einer Eingabemenge X ist dann folgendermaßen:

1. Erstelle k initiale Zentren $z_1, \ldots, z_k$ (z. B. randomisiert).
2. Erzeuge eine Partition $C_1, \ldots, C_k$ von X, so dass für alle $1 \leq i \leq k$ gilt:

$$f_d(x, z_i) = \min_{1 \leq j \leq k} f_d(x, z_j) \quad \text{für alle } x \in C_i \tag{3.2}$$

   Dabei soll $C_i \neq \emptyset$ für alle $1 \leq i \leq k$ gelten.
3. Berechne die z_i neu als Clusterzentren der C_i.
4. Beende, falls Gleichung (3.2) immer noch für alle $1 \leq i \leq k$ gilt, ansonsten springe zurück zu Schritt 2.

Es ist zu beachten, dass der k-means-Algorithmus nicht notwendigerweise terminiert, da die berechnete Partition nicht konvergieren muss. Dies hängt insbesondere von der verwendeten Abstandsfunktion ab. Ein weiterer Nachteil verbirgt sich im ersten Schritt. Dem Algorithmus müssen entweder k initiale Zentren mitgegeben werden oder er muss in die Lage versetzt werden, sich solche in sinnvoller Weise selbst zu erzeugen. Da das Ergebnis von der Wahl der Zentren zu Beginn abhängt, muss also dafür gesorgt werden, dass hier nicht bereits der Weg zu einem guten Ergebnis verbaut wird.

Was die Effizienz der Berechnung angeht, gibt es beweisbar schlechte Startbedingungen. In [Arthur und Vassilvitskii 2006] zeigen die Autoren ein Bad-Case-Szenario, in dem der k-means-Algorithmus $2^{\Omega(\sqrt{n})}$ Iterationen benötigt. Für die dort verwendete Abstandsfunktion bedeutet das aber, dass zumindest die Terminierung des Algorithmus gesichert ist.

3.2. Hierarchische Clustering-Algorithmen

Beim hierarchischen Clustering wird das k-Clustering-Problem für eine Menge X nicht nur für einen, sondern für alle möglichen Werte von k gelöst. Da wir keine

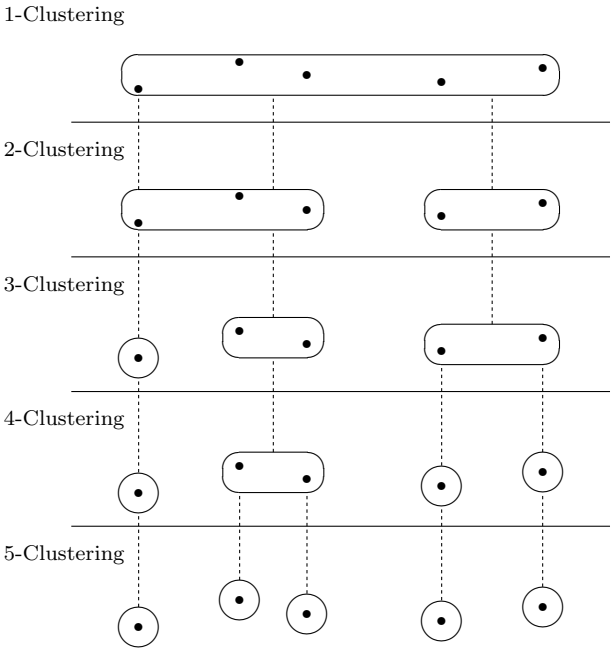

Abbildung 3.1.: Hierarchisches Clustering einer 5-elementigen Punktmenge

leeren Cluster zulassen, kann k alle Werte zwischen 1 und $|X|$ annehmen. Dabei sind die Cluster eines hierarchischen Clusterings ineinander verschachtelt, d. h. im $k+1$-Clustering sind alle Cluster bis auf zwei identisch zum k-Clustering (für $0 < k < |X|$). Die übrigen zwei Cluster des $k+1$-Clustering bilden zusammen den fehlenden Cluster des k-Clusterings. Abbildung 3.1 skizziert ein hierarchisches Clustering.

Wir betrachten zwei naheliegende Verfahren zur Berechnung eines hierarchischen Clusterings, einen agglomerativen und einen Divisive-Algorithmus. Agglomerative Algorithmen beginnen mit dem Clustering, in dem jeder Cluster genau ein Element enthält. Dann erzeugen sie das jeweils nächstkleinere Clustering durch die Vereinigung zweier Cluster, bis alle Elemente in einem großen Cluster verschmolzen sind. Dahingegen beginnen Divisive-Algorithmen mit dem 1-Clustering von X. Sie zerschlagen in jedem Schritt genau einen der Cluster in zwei neue, bis jeder Cluster nur noch ein Element enthält.

Man kann die beiden Verfahren auch zur Lösung des k-Clustering-Problems verwenden und die Berechnung abbrechen, sobald das k-Clustering berechnet ist. Man erhält dann Lösungen für Werte von 1 bis k beim Divisive-Algorithmus und für Werte von k bis $|X|$ beim agglomerativen Algorithmus.

3.2.1. Agglomeratives Clustering

Der einfache agglomerative Algorithmus, wie wir ihn jetzt vorstellen, approximiert das k-Clustering-Problem mit Zentren (Problem 2.4) für alle $1 \leq k \leq |X|$. Genau

wie der k-means-Algorithmus benötigt er eine Abstandsfunktion $f_d(\cdot, \cdot)$ (siehe Bemerkung 3.1) und eignet sich daher ebenfalls für Kostenfunktionen, die auf einem Abstandsmaß basieren. Den i-ten für ein k-Clustering berechneten Cluster bezeichnen wir im folgenden mit $C_{k,i}$.

Einfacher agglomerativer Algorithmus Sei $X = \{x_1, \ldots, x_n\}$ die zu clusternde Eingabemenge. Der einfache agglomerative Algorithmus arbeitet dann wie folgt:

- Erzeuge für jedes $x_i \in X$ einen Cluster $C_{n,i} = \{x_i\}$.
- Berechne $z_{n,i}$ als Zentrum von $C_{n,i}$ für $i = 1, \ldots, n$.
- Durchlaufe für Werte ℓ von $n-1$ bis 1 die folgenden Schritte:
  1. Finde Indizes $m_1 < m_2$, so dass gilt:
  $$f_d(z_{\ell+1,m_1}, z_{\ell+1,m_2}) = \min_{\substack{1 \le i,j \le \ell+1 \\ i \neq j}} f_d(z_{\ell+1,i}, z_{\ell+1,j})$$
  2. Setze $C_{\ell,m_1} := C_{\ell+1,m_1} \cup C_{\ell+1,m_2}$.
  3. Berechne z_{ℓ,m_1} als Zentrum von C_{ℓ,m_1}.
  4. Falls $m_2 \le \ell$, setze $C_{\ell,m_2} := C_{\ell+1,\ell+1}$ und $z_{\ell,m_2} := z_{\ell+1,\ell+1}$.
  5. Für alle $1 \le i \le \ell$ mit $i \neq m_1$ und $i \neq m_2$ setze $C_{\ell,i} := C_{\ell+1,i}$ und $z_{\ell,i} := z_{\ell+1,i}$.

Um das berechnete Clustering, wie in Problemdefinition 2.1 gefordert, als Abbildung zu erhalten, definieren wir noch

$$c_k : X \to \{1, \ldots, k\}, \quad c_k(x) \mapsto i \text{ mit } x \in C_{k,i}.$$

Wie bereits erwähnt, beginnt unser einfacher agglomerativer Algorithmus mit einem n-Clustering, in dem jeder Cluster aus einem Element der Eingabemenge besteht. Als nächstes werden zu diesen Clustern die Zentren berechnet. Dieser Schritt hängt natürlich von der vereinbarten Vorschrift zur Berechnung eines Clusterzentrums ab. Allerdings ist im Normalfall anzunehmen, dass das Zentrum eines 1-elementigen Clusters das Element selbst ist. In diesem Fall passiert in diesem Schritt nicht mehr als Namensgebung.

Anschließend wird in $n-1$ Durchläufen einer Schleife jeweils aus einem $\ell+1$-Clustering durch Vereinigung zweier Cluster ein ℓ-Clustering erzeugt. In Schritt 1 der Schleife werden die beiden Cluster gesucht, deren Zentren den geringsten Abstand zueinander haben. Die Vereinigung der beiden Cluster für das nächste Clustering findet dann in Schritt 2 statt. In Schritt 3 wird für den vereinigten Cluster das für die nächsten Schleifendurchläufe benötigte Clusterzentrum berechnet. Die Schritte 4 und 5 übernehmen schließlich nur die restlichen Cluster und deren Zentren unverändert ins nächste Clustering.

Der Vorteil des einfachen agglomerativen Algorithmus liegt in der Implementierbarkeit. Die einzigen Berechnungen, die durchgeführt werden, sind die Abstandsberechnungen zwischen den Zentren. Allerdings müssen in Schritt 1 $O(n^2)$ viele Abstände verglichen werden, wodurch sich eine Gesamtlaufzeit von $O(n^3)$ ergibt. Näheres zur Laufzeit kann bei [Day und Edelsbrunner 1984] nachgelesen werden. Dort wird auch ein agglomerativer Clustering-Algorithmus beschrieben, der nur eine Laufzeit von $O(n^2 \log n)$ besitzt.

Unser einfacher agglomerativer Algorithmus ist nur einer von vielen denkbaren agglomerativen Algorithmen. Andere agglomerative Algorithmen werden sich im Prinzip nur in Schritt 1, in dem die beiden zu verschmelzenden Cluster bestimmt werden, von ihm unterscheiden.

Der Einfachheit halber sprechen wir von nun an nur noch vom agglomerativen Algorithmus, meinen damit aber obigen einfachen agglomerativen Algorithmus.

3.2.2. Divisive-Clustering

In diesem Abschnitt legen wir uns auf die Kostenfunktion cost_{ZK} aus Problem 2.5 fest. Mit anderen Worten, wir betrachten das k-Clustering-Problem mit Zentrumskosten. Die Kosten eines Clusterings belaufen sich also auf die Summe der Abstände aller Elemente zu ihrem jeweiligen Clusterzentrum. Um die Notation zu vereinfachen, bezeichnen wir den Anteil eines Clusters $C \subseteq X$ an den Gesamtkosten mit $\text{cost}_{\text{ZK}}(C)$. Außerdem vereinbaren wir $\text{cost}_{\text{ZK}}(\emptyset) := 0$. Den i-ten für ein k-Clustering berechneten Cluster bezeichnen wir wieder mit $C_{k,i}$. Der folgende einfache Divisive-Algorithmus berechnet Approximationen für das k-Clustering-Problem mit Zentrumskosten (Problem 2.5) für alle $1 \leq k \leq |X|$.

Einfacher Divisive-Algorithmus Sei $X = \{x_1, \ldots, x_n\} \subseteq P$ die zu clusternde Eingabemenge. Der einfache Divisive-Algorithmus arbeitet nun wie folgt:

- Erzeuge einen Cluster $C_{1,1} = X$.
- Für Werte ℓ von 2 bis n führe folgende Anweisungen aus:
  - → Finde einen Index m, der die Funktion

    $$s(j) := \max_{x \in C_{\ell-1,j}} \left(\text{cost}_{\text{ZK}}(C_{\ell-1,j}) - \text{cost}_{\text{ZK}}(C_{\ell-1,j} \setminus \{x\}) + \text{cost}_{\text{ZK}}(\{x\})\right)$$

    über $\{1, \ldots, \ell - 1\}$ maximiert.
  - → Setze $C_{\ell,i} := C_{\ell-1,i}$ für alle $1 \leq i \leq \ell - 1$ und $C_{\ell,\ell} := \emptyset$.
  - → Solange ein $x \in C_{\ell,m}$ existiert mit

    $$\text{cost}_{\text{ZK}}(C_{\ell,m}) - \text{cost}_{\text{ZK}}(C_{\ell,m} \setminus \{x\}) + \text{cost}_{\text{ZK}}(C_{\ell,\ell} \cup \{x\}) - \text{cost}_{\text{ZK}}(C_{\ell,\ell}) > 0$$

    oder noch $C_{\ell,\ell} = \emptyset$ gilt, führe die folgenden beiden Schritte aus:

1. Ermittle einen Index i, so dass $x_i \in C_{\ell,m}$ gilt und x_i die Funktion

$$t(x) := \big(\text{cost}_{\text{ZK}}(C_{\ell,m}) - \text{cost}_{\text{ZK}}(C_{\ell,m} \setminus \{x\}) + \text{cost}_{\text{ZK}}(C_{\ell,\ell} \cup \{x\}) - \text{cost}_{\text{ZK}}(C_{\ell,\ell})\big)$$

   über $C_{\ell,m}$ maximiert.
2. Setze $C_{\ell,\ell} := C_{\ell,\ell} \cup \{x_i\}$ und $C_{\ell,m} := C_{\ell,m} \setminus \{x_i\}$.

Wie beim agglomerativen Algorithmus definieren wir

$$c_k : X \to \{1, \ldots, k\}, \quad c_k(x) \mapsto i \text{ mit } x \in C_{k,i},$$

um das berechnete Clustering, wie in Problemdefinition 2.1 gefordert, als Abbildung zu erhalten. Außerdem sprechen wir wie schon beim agglomerativen Algorithmus der Einfachheit halber von nun an nur noch vom Divisive-Algorithmus, meinen damit aber den einfachen Divisive-Algorithmus.

Die Berechnungen des Divisive-Algorithmus sind aufwändiger als beim agglomerativen Algorithmus, weil sie nicht nur auf der Anwendung einer Abstandsfunktion, sondern auf der Auswertung der Kostenfunktion beruhen. Der Divisive-Algorithmus beginnt, wie bereits angekündigt, mit dem 1-Clustering der Menge X. Von dort ausgehend wird $(n-1)$-mal durch Spaltung eines Clusters das jeweils nächste Clustering berechnet, bis sich jedes Element in einem eigenen Cluster befindet. Dabei wird zunächst der Cluster bestimmt, der zerteilt werden soll. Dazu wird für jedes Element $x \in X$ die Kostenersparnis berechnet, die auftritt, wenn x in einen eigenen Cluster verschoben wird und ein Cluster ausgewählt, der diese Kostenersparnis maximiert. Anschließend wird ein neuer leerer Cluster erzeugt und die Elemente des gewählten Clusters werden versuchsweise in den neuen Cluster verschoben, um die entstehende Kostenersparnis zu ermitteln. Solange noch ein Element existiert, das Kosten einspart, wenn es verschoben wird, wird das Element verschoben, das am meisten Kosten einspart. Dabei werden jedesmal neue Clusterzentren berechnet und Clusterkosten aufsummiert. Diese Operationen verstecken sich in den Auswertungen der cost_{ZK}-Funktion.

Für eine einfache Laufzeituntersuchung nehmen wir an, dass in jedem Schritt immer alle Elemente verschoben werden. Wir erhalten dann immer einen großen Cluster und den Rest als 1-elementige Cluster. Da immer nur das Element mit maximaler Kostenersparnis verschoben wird, haben wir in jedem Schritt der inneren Schleife $O(n)$ Vergleiche. Die innere Schleife selbst wird genau wie die äußere Schleife $O(n)$ oft durchlaufen. Wir erhalten also eine Gesamtlaufzeit von $O(n^3)$.

In [Guénoche u. a. 1991] wird ein Divisive-Algorithmus mit einer Laufzeit von nur $O(n^2 \log n)$ angegeben, allerdings für eine andere Kostenfunktion.

4. Analyse

4.1. Untersuchungen zur quadrierten L_2-Norm

Dieser Abschnitt beschäftigt sich mit einigen hilfreichen Aussagen über das k-Clustering-Problem mit Zentrumskosten und der L_2^2-Norm als Abstandsmaß. Die gewonnenen Erkenntnisse werden für die spätere Analyse des Divisive-Algorithmus benötigt.

Wählen wir den Schwerpunkt (siehe Definition 2.6) eines Clusters als Zentrum, dann benötigen wir nach dem folgenden Lemma zur Berechnung der Kostenersparnis beim Entfernen eines Punktes aus einem Cluster nur den Abstand des Punktes zum Zentrum und die Größe des Clusters.

Lemma 4.1 *Für einen beliebigen Cluster $C \subset \mathbb{R}^n$ und alle $x \in C$ gilt*

$$d\big(C, \operatorname{cog}(C)\big) \;-\; d\big(C \setminus \{x\}, \operatorname{cog}(C \setminus \{x\})\big) \;=\; \frac{|C|}{|C|-1}\, d(\operatorname{cog}(C), x)$$

mit der L_2^2-Norm als Abstandsmaß d.

Beweis Sei $C \subset \mathbb{R}^n$ mit $|C| \geq 2$ (für $|C| < 2$ ist nichts zu zeigen) und $x \in C$. Aus Definition 2.6 folgt dann

$$\operatorname{cog}(C) = \frac{1}{|C|}\Big((|C|-1)\operatorname{cog}(C \setminus \{x\}) + x\Big).$$

Wir lösen nach $\operatorname{cog}(C \setminus \{x\})$ auf und erhalten

$$\operatorname{cog}(C \setminus \{x\}) = \frac{1}{|C|-1}\Big(|C|\operatorname{cog}(C) - x\Big). \tag{4.1}$$

Die Kosten von C lassen sich in folgende Summe aufspalten:

$$d\big(C, \operatorname{cog}(C)\big) \;=\; d\big(C \setminus \{x\}, \operatorname{cog}(C)\big) \;+\; d\big(x, \operatorname{cog}(C)\big) \tag{4.2}$$

Aus Lemma 2.7 folgt dann für den ersten Summanden von (4.2):

$$\begin{aligned} & d\big(C \setminus \{x\}, \operatorname{cog}(C)\big) \\ = \;& d\big(C \setminus \{x\}, \operatorname{cog}(C \setminus \{x\})\big) \;+\; (|C|-1)\, d\big(\operatorname{cog}(C \setminus \{x\}), \operatorname{cog}(C)\big) \end{aligned} \tag{4.3}$$

Mithilfe von (4.1) schreiben wir den zweiten Summanden von (4.3) um:

$$\begin{aligned}
& (|C|-1)\, d\big(\mathrm{cog}(C \setminus \{x\}), \mathrm{cog}(C)\big) \\
\overset{(4.1)}{=}\ & (|C|-1)\, d\left(\frac{|C|\,\mathrm{cog}(C) - x}{|C|-1}, \mathrm{cog}(C)\right) \\
=\ & (|C|-1) \left\| \frac{1}{|C|-1} \big(\mathrm{cog}(C) - x\big) \right\|^2 \\
=\ & \frac{1}{|C|-1} \big\|\mathrm{cog}(C) - x\big\|^2 \\
=\ & \frac{1}{|C|-1}\, d(\mathrm{cog}(C), x)
\end{aligned}$$

Durch Einsetzen in (4.3) und (4.2) erhalten wir schließlich:

$$\begin{aligned}
& d\big(C, \mathrm{cog}(C)\big) \\
=\ & d\big(C \setminus \{x\}, \mathrm{cog}(C \setminus \{x\})\big) \ +\ \frac{1}{|C|-1}\, d(\mathrm{cog}(C), x) \ +\ d\big(x, \mathrm{cog}(C)\big) \\
=\ & d\big(C \setminus \{x\}, \mathrm{cog}(C \setminus \{x\})\big) \ +\ \frac{|C|}{|C|-1}\, d(\mathrm{cog}(C), x)
\end{aligned}$$

□

Korrolar 4.2 *Für einen beliebigen Cluster $C \subset \mathbb{R}^n$, der L_2^2-Norm als Abstandsmaß d und $x_0 \in C$ gilt:*

$$\begin{aligned}
& d(C \setminus \{x_0\}, \mathrm{cog}(C \setminus \{x_0\})) = \min_{x \in C} d(C \setminus \{x\}, \mathrm{cog}(C \setminus \{x\})) \\
\iff\quad & d(x_0, \mathrm{cog}(C)) = \max_{x \in C} d(x, \mathrm{cog}(C))
\end{aligned}$$

Das bedeutet, wir erreichen beim Entfernen eines Elementes aus dem Cluster dann die größte Kostenersparnis, wenn wir ein Element mit größtem Abstand zum Zentrum entfernen.

Als nächstes betrachten wir die Veränderung der Gesamtkosten des Clusterings, wenn wir einen Punkt von einem Cluster in einen anderen verschieben.

Lemma 4.3 *Seien $C_1, C_2 \subset \mathbb{R}^n$ mit $|C_1| > 1$, $|C_2| > 0$ und sei $x \in C_1$. Dann beträgt die Veränderung δ der Gesamtkosten des Clusterings beim Verschieben von x nach C_2:*

$$\delta \ =\ \frac{|C_2|+1}{|C_2|}\, d(\mathrm{cog}(C_2 \cup \{x\}, x) \ -\ \frac{|C_1|}{|C_1|-1}\, d(\mathrm{cog}(C_1), x)$$

Beweis Für die Differenz δ zwischen den Kosten nach der Verschiebung und denen

davor gilt:

$$\begin{aligned}\delta = &\Big(d\big(C_1 \setminus \{x\}, \mathrm{cog}(C_1 \setminus \{x\})\big) + d\big(C_2 \cup \{x\}, \mathrm{cog}(C_2 \cup \{x\})\big)\Big) \\ &-\Big(d\big(C_1, \mathrm{cog}(C_1)\big) + d\big(C_2, \mathrm{cog}(C_2)\big)\Big) \\ = &\Big(d\big(C_2 \cup \{x\}, \mathrm{cog}(C_2 \cup \{x\})\big) - d\big(C_2, \mathrm{cog}(C_2)\big)\Big) \\ &-\Big(d\big(C_1, \mathrm{cog}(C_1)\big) - d\big(C_1 \setminus \{x\}, \mathrm{cog}(C_1 \setminus \{x\})\big)\Big)\end{aligned}$$

Die Behauptung folgt dann mit Lemma 4.1. □

4.2. Clustering-Algorithmen im Vergleich

Wir schauen uns nun die drei Algorithmen aus Kapitel 3 bei der Arbeit an. Die Implementierung aus Anhang A.1 werden wir dabei nutzen, um die Ausgaben der Algorithmen zu vergleichen. Dabei geht es uns in erster Linie um die Clusteringkosten. Um auch die Laufzeit der Algorithmen ernsthaft miteinander vergleichen zu können, müssten wir auch sicherstellen, dass sie gleich effizient implementiert[1] sind. Daher betrachten wir die Laufzeiten nur am Rande.

Als Probenraum wählen wir einen gewichteten Wahrscheinlichkeits-Simplex. Als Grundlage dient der Wahrscheinlichkeits-Simplex

$$\Delta_{16} := \left\{ p = (p_1, \ldots, p_{16}) \in \mathbb{R}^{16} \;\middle|\; \begin{array}{c} p_i \geq 0 \text{ für } i = \{1, \ldots, 16\} \\ \text{und } p_1 + p_2 + \ldots + p_{16} = 1 \end{array} \right\}.$$

Was es mit der Gewichtung auf sich hat, wird am Ende des nächsten Abschnitts erklärt.

4.2.1. Markov-Ketten zur Erzeugung von Wahrscheinlichkeitsverteilungen

Um passende Wahrscheinlichkeitsverteilungen zu erhalten, betrachten wir eine beliebige Eingabedatei als Folge von Nibbles (=halbe Bytes). Ein Nibble besteht aus 4 Bits und kann 16 verschiedene Werte annehmen. Δ_{16} ist also die Menge aller möglichen Wahrscheinlichkeitsverteilungen über dem Nibble-Alphabet.

Die zu clusternden Wahrscheinlichkeitsverteilungen erhalten wir nun aus der sogenannten Markov-Kette zweiter Ordnung. Eine genauere Auseinandersetzung mit diesem Thema kann bei [Hoppe 2003] nachgelesen werden. Für unsere Zwecke wollen wir die Erzeugung der Verteilungen nur kurz beschreiben.

[1] Der Bedeutung der Eigenschaft ‚gleich effizient implementiert' auf den Grund zu gehen, würde uns zu weit von unserem eigentlichen Thema wegführen.

Wir lesen die gesamte Eingabedatei ein und zählen dabei die Häufigkeiten des Auftretens jedes möglichen Nibble-Werts. Dies tun wir allerdings in Abhängigkeit von den beiden zuletzt gelesenen Nibble-Werten (2 Nibble = 1 Byte). Wir erhalten so eine Menge von Häufigkeiten $h_{a,b}$ mit $a \in \{0, \ldots, 255\}$ und $b \in \{0, \ldots, 15\}$. Dabei bezeichnet $h_{a,b}$ die Häufigkeit des Auftretens von b im Zustand, in dem als letztes die Bits von a gelesen wurden. Wir nennen diesen Zustand im folgenden S_a.

Für alle Werte von a mit

$$H_a := \sum_{i=0}^{15} h_{a,i} > 0 \tag{4.4}$$

erzeugen wir nun eine Wahrscheinlichkeitsverteilung $p_a \in \Delta_{16}$ mit

$$p_a \;=\; (p_{a,0}, \ldots, p_{a,15}),$$

wobei für $i \in \{0, \ldots, 15\}$ gilt:

$$p_{a,i} = \frac{h_{a,i}}{H_a}$$

Wir erhalten so eine Menge

$$P \;=\; \{p_a \,|\, a \text{ erfüllt Gleichung (4.4)}\}$$

von höchstens 256 Wahrscheinlichkeitsverteilungen. p_a ist dabei die Wahrscheinlichkeitsverteilung für das nächste Symbol im Zustand S_a unseres Markov-Modells. H_a beschreibt die Häufigkeit, mit der wir uns bei der Erzeugung der Wahrscheinlichkeitsverteilungen im Zustand S_a befunden haben. Die Summe der H_a nennen wir

$$H := \sum_{p_a \in P} H_a.$$

Dann erhalten wir mit

$$w_a := \frac{H_a}{H}$$

eine Wahrscheinlichkeitsverteilung über die Zustände S_a. Die w_a sollen nun die Gewichte unserer Wahrscheinlichkeitsverteilungen p_a sein. Das ergibt den gewichteten Wahrscheinlichkeits-Simplex

$$X \;:=\; \{(p_a, w_a) | p_a \in P\},$$

der uns als Eingabemenge für die Clustering-Algorithmen dient. Für $x \in X$ führen wir die Schreibweise

$$x \;=\; \big(v(x), w(x)\big)$$

ein.

4.2.2. Berechnung der Clusterzentren

Als Clusterzentrum wählen wir das gewichtete Mittel der gewichteten Wahrscheinlichkeitsverteilungen. Für einen Cluster $C \subseteq X$ berechnet sich das Zentrum $z(C)$ folgendermaßen:

$$z(C) := \left(\frac{\sum_{x \in C} w(x) \cdot v(x)}{\sum_{x \in C} w(x)}, \sum_{x \in C} w(x) \right)$$

Die Menge aller möglichen Clusterzentren über X nennen wir Z mit

$$Z := z(\mathcal{P}(X) \setminus \emptyset) = \{x \in \Delta_{16} \times \mathbb{R} | x = z(C) \text{ für ein } C \subseteq X, C \neq \emptyset\}.$$

Wegen $z(\{x\}) = x$ folgt

$$X \subseteq Z.$$

4.2.3. Die Kostenfunktion

Wir wählen die KLD (siehe Definition 2.13) als Abstandsmaß, das unserer Kostenfunktion zugrunde liegt. Als Kostenfunktion wählen wir die Summe der gewichteten KLDs zu den Clusterzentren:

$$\begin{aligned} \text{cost}(c) &= \sum_{x \in X} w(x) \cdot D\left(v(x) \,|\, v\left(z(C_{c(x)})\right)\right) \\ &= \sum_{i=1}^{k} \sum_{\substack{x \in X \\ c(x)=i}} w(x) \cdot D\left(v(x) \,|\, v\big(z(C_i)\big)\right), \end{aligned}$$

wobei $C_i = \{x \in X | c(x) = i\}$ für $i = 1, \ldots, k$ gilt und $c : X \to \{1, \ldots, k\}$ das berechnete k-Clustering beschreibt. Die Gesamtkosten eines Clusterings entsprechen also dem Kompressionsverlust, der entsteht, wenn jede Verteilung durch die Verteilung ihres Clusterzentrums ersetzt wird (vergl. Abschnitt 2.2.3).

4.2.4. Verwendete Abstandsfunktion

Unsere Abstandsfunktion (siehe Bemerkung 3.1) basiert genau wie die Kostenfunktion auf der KLD. Als Abstandsfunktion wählen wir die gewichtete KLD zum gewichteten Mittel der Verteilungen. Seien $C_1, C_2 \subseteq X$ mit $C_1, C_2 \neq \emptyset$. Um den Abstand zwischen $z(C_1) = (p_i, w_i) \in Z$ und $z(C_2) = (p_j, w_j) \in Z$ zu bestimmen, berechnen wir zunächst das gewichtete Mittel

$$z(C_1 \cup C_2) = (q, w) := \left(\frac{w_i \cdot p_i + w_j \cdot p_j}{w_i + w_j}, w_i + w_j \right).$$

Unsere Abstandsfunktion berechnet nun die Summe

$$d\big((p_i, w_i), (p_j, w_j)\big) \;=\; w_i \cdot D(p_i|q) \;+\; w_j \cdot D(p_j|q)$$

der gewichteten KLDs von p_i und p_j zu q. Dies entspricht genau dem Kompressionsverlust, der entsteht, wenn die Verteilungen $\bigcup_{x \in C_1 \cup C_2} v(x)$ durch q ersetzt würden (vergl. Abschnitt 2.2.3). Die KLD jeder Verteilung $v(x)$ geht dabei mit ihrem Gewicht $w(x)$ ein, wobei $w(x)$ gerade die Wahrscheinlichkeit des Auftretens in unserem Markov-Modell aus Abschnitt 4.2.1 ist.

4.2.5. Eingabedateien für das Markov-Modell

Als Eingabedateien für die Erzeugung des gewichteten Wahrscheinlichkeits-Simplex (wie in Abschnitt 4.2.1 beschrieben) verwenden wir drei vom Inhalt möglichst unterschiedliche Dateien.

Die erste Datei ist ein deutscher Text. Um repräsentative Verteilungen für einen Text in deutscher Sprache zu erhalten, wird ein möglichst langer Text benötigt. Dieser wurde in Form einer frei verfügbaren deutschen Bibelübersetzung[2] gefunden. Die ASCII-Datei hat eine Größe von 4.788.108 Bytes. Aus dem Markov-Modell erhalten wir 55 Wahrscheinlichkeitsverteilungen.

Die zweite Datei ist eine Bilddatei. Es handelt sich um eine Luftaufnahme vom Campus der Universität Paderborn[3]. Die JPEG-Datei hat eine Größe von 3.382.578 Bytes. Aus dem Markov-Modell erhalten wir hier 90 Wahrscheinlichkeitsverteilungen.

Die dritte und letzte Datei ist ein ausführbares Programm in Form eines Linux-Binarys. Es handelt sich um das im Anhang A.1 erwähnte Testprogramm. Die Datei hat eine Größe von 554.203 Bytes. Aus dem Markov-Modell erhalten wir ebenfalls 90 Wahrscheinlichkeitsverteilungen.

4.2.6. Ergebnis der Untersuchungen

Auch wenn unser Fokus nicht auf den Algorithmen-Laufzeiten liegt, werden wir kurz darauf eingehen. Daher sei an dieser Stelle angemerkt, dass sämtliche Testläufe auf einem Testsystem mit folgender Konfiguration durchgeführt wurden:

- Intel® Pentium® M Prozessor 1700MHz
- 512 MB Hauptspeicher
- Gentoo Linux Betriebssystem
- gcc C++ Compiler Version 4.1.1

[2] Unrevidierte Elberfelder Übersetzung, R. Brockhaus, 1905, verfügbar unter `http://www.kahal.de/bibel_elb_bibel.zip`

[3] verfügbar unter `http://www.uni-paderborn.de/fotoarchiv/images2/20000155.jpg`

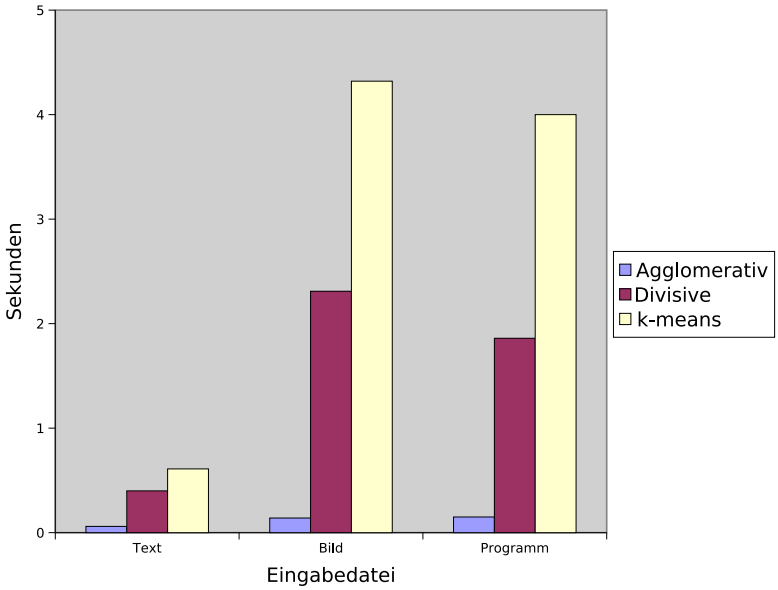

Abbildung 4.1.: Laufzeiten untersuchter Clustering-Algorithmen

Wie bereits in Kapitel 3 festgestellt, ist der Divisive-Algorithmus rechenintensiver als der agglomerative Algorithmus. Dies wird durch unsere Implementierung bestätigt. Für jede der drei Eingabedateien waren die Berechnungen des agglomerativen Algorithmus um ein Vielfaches schneller, wie man an Abbildung 4.1 ablesen kann. Zum k-means Algorithmus ist noch anzumerken, dass für jeden möglichen Wert von k einzeln ein k-Clustering berechnet und die Laufzeiten aufsummiert wurden. Dadurch sind die Laufzeiten mit denen der beiden hierarchischen Algorithmen, die ja auch alle möglichen k-Clusterings berechnen, einigermaßen vergleichbar. Auf eine Wiederverwendung der dabei berechneten Clusterings für den jeweils nächsten Durchlauf, beispielsweise zur Optimierung der initialen Clusterzentren (siehe Abschnitt 3.1.1), wurde bewusst verzichtet. Der k-means Algorithmus wie wir ihn beschrieben haben, löst nun einmal das k-Clustering-Problem nur für einen einzelnen Wert von k. Die initialen Clusterzentren werden in unserer Implementierung dadurch erzeugt, dass alle Elemente der Eingabe gleichmäßig[4] auf initiale Cluster verteilt werden, von denen dann die Zentren berechnet werden.

Die Abhängigkeit des k-means Algorithmus von der Wahl der initialen Clusterzentren lässt sich gut an Abbildung 4.2 ablesen. Dort sind die Kosten der berechneten Clusterings für die Textdatei von allen drei untersuchten Algorithmen dargestellt. Man erkennt, dass beim k-means Algorithmus die Kosten des k-Clusterings für steigene Werte von k nicht monoton fallend sind. Dies liegt daran, dass der Algorithmus jedes Clustering von Grund auf neu berechnet und dabei mit ungünstigen initialen Clusterzentren beginnen kann. Betrachtet man die Kosten für die

[4] Alle Elemente werden in der Eingabereihenfolge durch Abzählen bis k aufgeteilt.

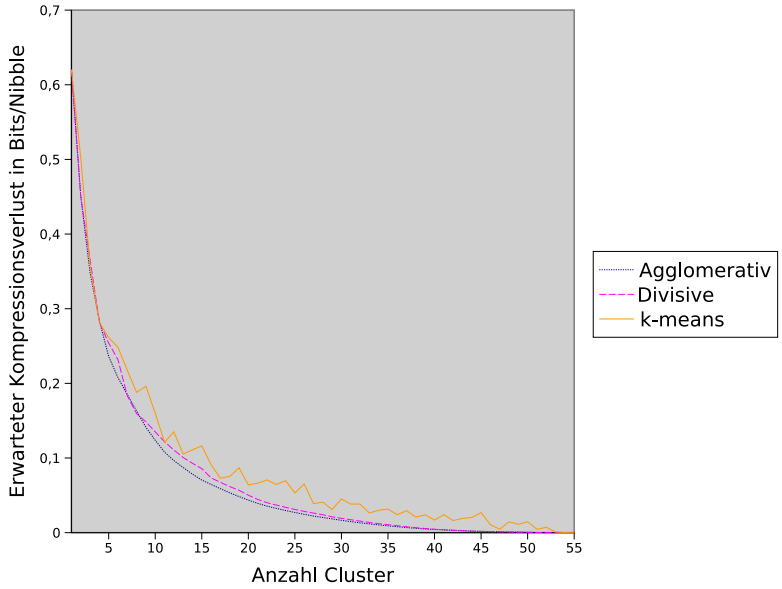

Abbildung 4.2.: Erwarteter Kompressionsverlust für die Textdatei

Bilddatei in Abbildung 4.3, fällt dieser Sachverhalt noch mehr auf. Dort sind beim k-means Algorithmus mehrere große Spitzen der Kostenfunktion zu beobachten.

Bei den hierarchischen Algorithmen sieht dies erwartungsgemäß anders aus. Beim agglomerativen Algorithmus können durch die Vereinigung zweier Cluster keine Kosten gespart werden, da mit dem Schwerpunkt als Clusterzentrum beide Cluster vor der Vereinigung bereits minimale Kosten verursachen. Das gleiche gilt entsprechend umgekehrt für den Divisive-Algorithmus. Die Ergebnisse für die Programmdatei in Abbildung 4.4 bestätigen die Beobachtungen, liefern aber keine neuen Einsichten.

Die wichtigste Erkenntnis, die man aus den Abbildungen 4.1 bis 4.4 ziehen kann, betrifft allerdings den Vergleich zwischen agglomerativem und Divisive-Algorithmus. Es handelt sich dabei um die Tatsache, dass die Kosten beim agglomerativen Algorithmus fast für jede Anzahl an Clustern unter den Kosten vom Divisive-Algorithmus liegen. Das bedeutet, dass der einfachste Algorithmus fast durchgängig die besten Ergebnisse erzielt, und das bei der geringsten Laufzeit. Eine Antwort auf die Frage, warum dies so ist, hat sich im Rahmen dieser Arbeit nicht ergeben, ist aber sicherlich einer genaueren Betrachtung wert.

4.3. Bad-Case-Szenario für den agglomerativen Algorithmus

In diesem Abschnitt konstruieren wir zwei Punktmengen, auf denen der agglomerative Clustering-Algorithmus (siehe Abschnitt 3.2.1) ein Ergebnis liefert, das

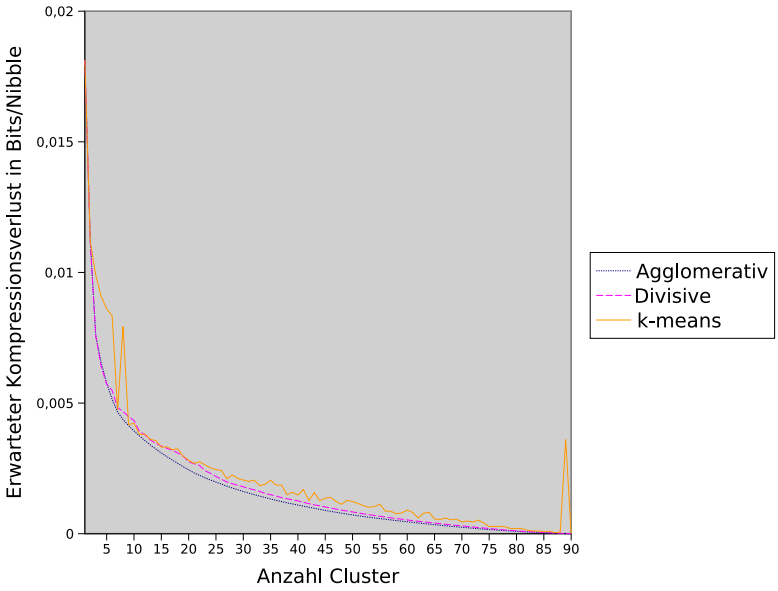

Abbildung 4.3.: Erwarteter Kompressionsverlust für die Bilddatei

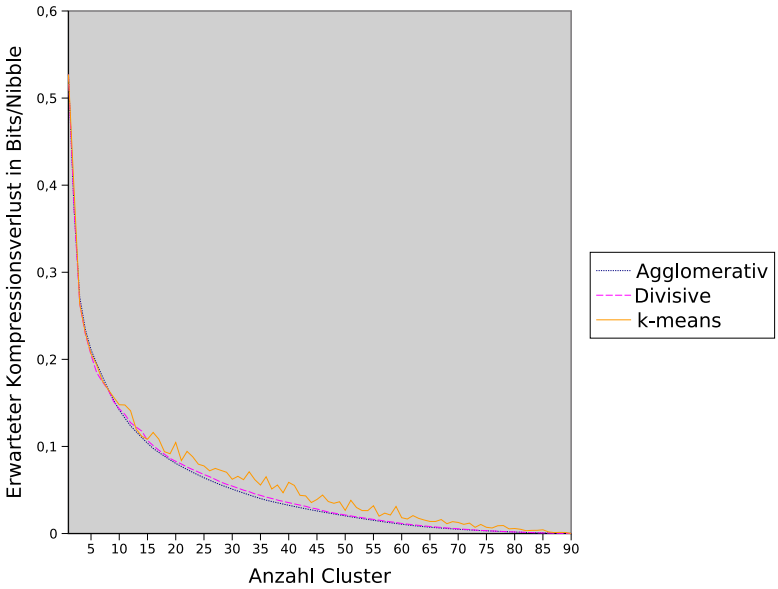

Abbildung 4.4.: Erwarteter Kompressionsverlust für die Programmdatei

wesentlich vom Optimum entfernt ist. In Abschnitt 4.3.1 betrachten wir dabei eine Mahalanobis-Distanz und in Abschnitt 4.3.2 die KLD sowohl als Abstandsmaß als auch als Abstandsfunktion (siehe Bemerkung 3.1). Die Grundidee des Beweises basiert auf einem Bad-Case-Szenario für den agglomerativen Algorithmus mit der L_1-Norm bei [Dasgupta und Long 2005].

4.3.1. Diagonale Mahalanobis-Distanz

Ziel dieses Abschnitts ist es, zu zeigen, dass das agglomerative Clusteringverfahren für Punkte des $\mathbb{R}^n$ um den Faktor n schlechter sein kann als die optimale Lösung. Als Abstandsmaß d auf dem $\mathbb{R}^n$ wählen wir die diagonale Mahalanobis-Distanz aus Definition 2.10. Als Kostenfunktion wählen wir den größten Durchmesser eines Clusters unter dem Abstandsmaß d. Wir untersuchen also das Durchmesser minimierende k-Clustering-Problem (Problem 2.3). Da wir aber den agglomerativen Algorithmus in Kapitel 3 für das k-Clustering-Problem mit Zentren (Problem 2.4) angegeben haben, müssen wir festlegen, wie sich die Clusterzentren errechnen. Wir wählen dazu den Schwerpunkt eines Clusters (siehe Definition 2.6) als sein Zentrum.

Wir werden nun nach einigen Vorüberlegungen den folgenden Satz beweisen.

Satz 4.4 *Für das Durchmesser minimierende k-Clustering-Problem mit der diagonalen Mahalanobis-Distanz als Abstandsmaß existiert für $n \in \mathbb{N}$ eine Punktmenge der Größe 2^{n+1} im $\mathbb{R}^n$, so dass der agglomerative Clustering-Algorithmus mit dem Schwerpunkt als Clusterzentrum ein 2^n-Clustering liefert, das beliebig nahe am n-fachen der optimalen Lösung liegt.*

Wir wählen ein $\epsilon \in \mathbb{R}$ mit $0 < \epsilon < \frac{1}{n}$ und beginnen die Konstruktion unserer Punktmenge mit der Definition einiger 2-elementiger Mengen:

$$B_1 := \frac{1}{\sqrt{c_1}}\{-3, +3\},$$

für $j = 2, \ldots, n$ sei

$$B_j := \frac{1}{\sqrt{c_j}}\{-1, +1\}$$

und für $j = 1, \ldots, n$ sei

$$C_j := \frac{1}{\sqrt{c_j}}\{-(1 - j\epsilon), +(1 - j\epsilon)\}.$$

Dann erhalten wir durch

$$P_1 := B_1 \times \ldots \times B_n \quad \text{und} \quad P_2 := C_1 \times \ldots \times C_n$$

zwei 2^n-elementige Teilmengen, die jeweils die Eckpunkte eines n-dimensionalen Quaders beschreiben. Mit den entsprechenden c_j sind die Seitenlängen des durch

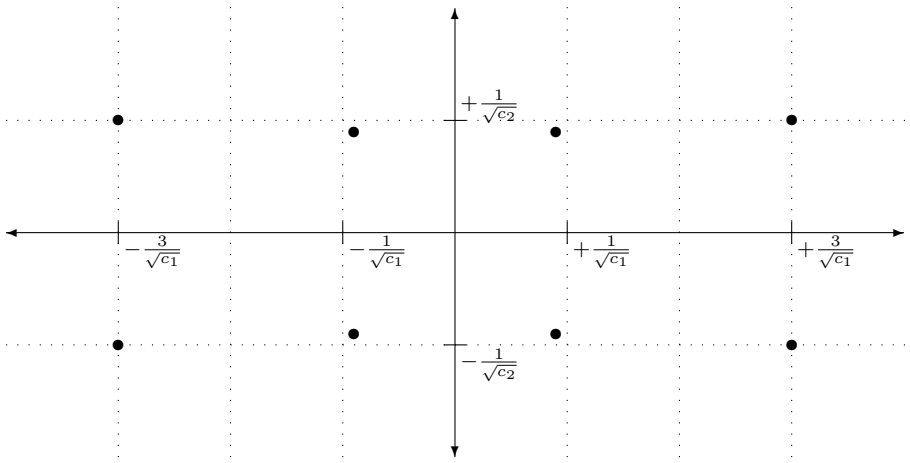

Abbildung 4.5.: Die Punktmengenkonstruktion im Fall $n = 2$

P_1 beschriebenen Quaders gleich $\frac{6}{\sqrt{c_1}}$ bzw. $\frac{2}{\sqrt{c_j}}$, während sie für P_2 alle etwas kleiner als $\frac{2}{\sqrt{c_j}}$ sind. Unsere gesamte Punktmenge P besteht nun aus der Vereinigung beider Mengen, also $P := P_1 \cup P_2$. Abbildung 4.5 skizziert P für den Fall $n = 2$.

Als nächstes zeigen wir einige Eigenschaften von P bezüglich des Abstandsmaßes d.

Lemma 4.5 *Für $x, y \in P_1$ mit $x \neq y$ gilt $d(x, y) \geq 4$.*

Beweis Aus $x \neq y$ folgt, dass ein j existiert mit $x_j \neq y_j$. Im Fall $j = 1$ gilt dann $|x_j - y_j| = \frac{6}{\sqrt{c_j}}$ und für $1 < j \leq n$ gilt $|x_j - y_j| = \frac{2}{\sqrt{c_j}}$. Insgesamt folgt also für alle j: $|x_j - y_j| \geq \frac{2}{\sqrt{c_j}}$. Schließlich gilt wegen $c_i > 0$ für $i = 1, \ldots, n$:

$$d(x, y) = \sum_{i=1}^{n} c_i (x_i - y_i)^2 \geq c_j (x_j - y_j)^2 \geq c_j \left(\frac{2}{\sqrt{c_j}} \right)^2 = 4$$

□

Lemma 4.6 *Für $x \in P_1$ und $y \in P_2$ gilt $d(x, y) > 4$.*

Beweis Denjenigen Punkt aus P_2, dessen Vorzeichen in allen Koordinaten mit denen von x übereinstimmt, nennen wir u. Aus der Konstruktion von P folgt, dass u von allen Punkten aus P_2 in jeder Koordinate den geringsten Abstand zu x hat und somit $d(x, u)$ minimiert. Es reicht also zu zeigen: $d(x, u) > 4$. Für den Unterschied von x und u in der ersten Koordinate gilt

$$|x_1 - u_1| = \frac{2 + \epsilon}{\sqrt{c_1}}.$$

Wegen $c_i > 0$ für $i = 1, \ldots, n$ gilt

$$d(x,y) = \sum_{i=1}^{n} c_i(x_i - y_i)^2 \geq c_1(x_1 - y_1)^2 = c_1 \left(\frac{2+\epsilon}{\sqrt{c_1}} \right)^2 > 4.$$

□

Lemma 4.7 *Für $x, y \in P_2$, wobei sich x und y an genau einer Koordinate j unterscheiden, gilt*

$$d(x,y) = 4(1 - j\epsilon)^2 < 4.$$

Beweis Sei j die Koordinate, an der sich x und y unterscheiden, dann gilt

$$|x_j - y_j| = \frac{2(1-j\epsilon)}{\sqrt{c_j}}.$$

Daraus folgt dann

$$d(x,y) = \sum_{i=1}^{n} c_i(x_i - y_i)^2 = c_j(x_j - y_j)^2 = c_j \left(\frac{2(1-j\epsilon)}{\sqrt{c_j}} \right)^2 = 4(1-j\epsilon)^2.$$

□

Lemma 4.8 *Für $x, y \in P_2$, wobei sich x und y an der j-ten Koordinate unterscheiden, gilt*

$$d(x,y) \geq 4(1 - j\epsilon)^2.$$

Beweis Es gilt

$$|x_j - y_j| = \frac{2(1-j\epsilon)}{\sqrt{c_j}}.$$

Es folgt wegen $c_i > 0$ für $i = 1, \ldots, n$:

$$d(x,y) = \sum_{i=1}^{n} c_i(x_i - y_i)^2 \geq c_j(x_j - y_j)^2 = c_j \left(\frac{2(1-j\epsilon)}{\sqrt{c_j}} \right)^2 = 4(1-j\epsilon)^2$$

□

Betrachten wir nun das Verhalten des agglomerativen Clusteringalgorithmus mit unserer Punktmenge P als Eingabe. Beginnend mit 2^{n+1} Clustern, die jeweils genau einen Punkt aus P enthalten, werden als erstes zwei Punkte zu einem Cluster vereinigt, die bezüglich d einen minimalen Abstand haben. Lemma 4.5 sagt aus, dass je zwei Punkte aus der Menge P_1 einen Abstand größer als 4 haben. Nach Lemma 4.6 gilt dies auch für Punktepaare, bei denen ein Punkt aus der Menge P_1 und ein Punkt aus der Menge P_2 stammt. Lemma 4.7 zeigt die Existenz von Punktepaaren aus P_2, die einen Abstand kleiner als 4 aufweisen. Zusammen mit Lemma 4.8 folgt, dass die Punktepaare (x, y) aus P_2, die sich nur in der letzten

Koordinate unterscheiden, d über P minimieren. Davon gibt es 2^{n-1} viele und alle haben nach Lemma 4.7 einen Abstand von $4(1-n\epsilon)^2$. Wir erhalten als Cluster-Zentrum einen Punkt $w = (x_1, \ldots, x_{n-1}, 0) = (y_1, \ldots, y_{n-1}, 0)$. Für w gilt analog zum Beweis von Lemma 4.6, dass für alle $x \in P_1$ gilt: $d(x, w) > 4$. Außerdem unterscheidet sich w zu allen übriggebliebenen Punkten $x \in P_2$ in einer Koordinate j mit $j < n$ und es folgt wie im Beweis von Lemma 4.8: $d(x, w) \geq 4(1-j\epsilon)^2$. Daher werden in den weiteren Iterationen des Algorithmus zunächst alle anderen Punktepaare aus P_2 vereint, die sich nur in der letzten Koordinate unterscheiden. So werden zunächst 2^{n-1} Cluster mit den Cluster-Zentren $Z = C_1 \times \ldots \times C_{n-1} \times \{0\}$ erzeugt.

Analog zum Zeitpunkt vor der ersten Iteration sind die Punkte bzw. Cluster-Zentren mit minimalem Abstand bezüglich d nun die 2^{n-2} Punkte aus Z, die sich nur in der $(n-1)$-ten Koordinate unterscheiden. Der Algorithmus fährt entsprechend fort, bis nach $2^n - 1$ Iterationen alle Punkte aus der Menge P_2 zu einem Cluster zusammengefasst wurden, während die Punkte aus der Menge P_1 noch jeder ein Cluster für sich beanspruchen. Eine Iteration später erhalten wir schließlich ein 2^n-Clustering, dessen Kosten dann natürlich mindestens so groß sind wie die des $2^n - 1$-Clusterings.

Beweis (Satz 4.4) Die Kosten des vom agglomerativen Algorithmus erzeugten 2^n-Clusterings sind mindestens gleich dem bezüglich d größten Abstand zweier Punkte aus P_2. Der Abstand zweier beliebiger Punkte aus P_2 ist also eine untere Schranke für diese Kosten. Wir wählen die beiden Punkte

$$x = \left(-\frac{1-\epsilon}{\sqrt{c_1}}, -\frac{1-2\epsilon}{\sqrt{c_2}}, \ldots, -\frac{1-n\epsilon}{\sqrt{c_n}}\right) \in P_2$$
$$\text{und} \quad y = \left(+\frac{1-\epsilon}{\sqrt{c_1}}, +\frac{1-2\epsilon}{\sqrt{c_2}}, \ldots, +\frac{1-n\epsilon}{\sqrt{c_n}}\right) \in P_2.$$

Dann gilt für die Kosten K_{agglo} des berechneten 2^n-Clusterings:

$$\begin{aligned} K_{\text{agglo}} &\geq d(x, y) \\ &= \sum_{i=1}^{n} c_i \left(\frac{2(1-j\epsilon)}{\sqrt{c_j}}\right)^2 = \sum_{i=1}^{n} 4(1-j\epsilon)^2 \geq 4\sum_{i=1}^{n}(1-n\epsilon)^2 \\ &= 4n\left(1 - 2n\epsilon + (n\epsilon)^2\right) > 4n - 8n^2\epsilon \end{aligned}$$

Ein weitaus besseres 2^n-Clustering ergibt sich aber, wenn wir jeweils einen Punkt $x \in P_1$ mit dem Punkt aus P_2 zusammenlegen, der in allen Koordinaten das gleiche

Vorzeichen hat wie x. Die Kosten K_{better} dieses Clusterings belaufen sich dann auf:

$$\begin{aligned} K_{\text{better}} &= c_1 \left(\frac{2+\epsilon}{\sqrt{c_1}}\right)^2 + \sum_{i=2}^{n} c_i \left(\frac{j\epsilon}{\sqrt{c_i}}\right)^2 = (2+\epsilon)^2 + \sum_{i=2}^{n} (j\epsilon)^2 \\ &= 4 + 4\epsilon + \epsilon^2 + \sum_{i=2}^{n} (j\epsilon)^2 = 4 + 4\epsilon + \sum_{i=1}^{n} (j\epsilon)^2 \\ &< 4 + 4\epsilon + \sum_{i=1}^{n} (n\epsilon)^2 = 4 + 4\epsilon + n^3\epsilon^2 \end{aligned}$$

Für den Approximationsfaktor des agglomerativen Clustering-Algorithmus gilt also

$$\frac{K_{\text{agglo}}}{K_{\text{better}}} > \frac{4n - 8n^2\epsilon}{4 + 4\epsilon + n^3\epsilon^2}.$$

Wenn wir ϵ klein genug wählen, können wir unsere untere Schranke für den Approximationsfaktor beliebig nahe an n heranbringen. □

4.3.2. Kullback-Leibler-Divergenz

In diesem Abschnitt wollen wir die Konstruktion aus dem Abschnitt 4.3.1 anpassen, so dass sich das Ergebnis auf ein weiteres Abstandsmaß übertragen lässt. Lemma 2.15 sagt aus, dass die KLD (siehe Definition 2.14) umso enger durch die L_2^2-Norm beschränkt werden kann, je größer die Koordinaten der beiden Punkte sind. Die L_2^2-Norm entspricht aber gerade dem Abstandsmaß d aus Definition 2.10 mit der Einheitsmatrix I_n als Matrix D. In diesem Abschnitt verfolgen wir die Idee, die Punktkonstruktion aus dem letzten Abschnitt um eine positive Konstante α entlang der Koordinatenachsen zu verschieben. Mit Hilfe von Lemma 2.15 können wir dann zeigen, dass der agglomerative Clustering-Algorithmus mit dem neuen Abstandsmaß für ein geeignetes α immer noch das gleiche Verhalten aufweist.

Wir konstruieren unsere Punktmenge nun fast genauso wie in Abschnitt 4.3.1, verschieben aber alle Koordinaten um eine Konstante α. Außerdem setzen wir die Werte $c_1, \ldots, c_n$ alle auf den Wert 1.

Wir wählen wieder ein $\epsilon \in \mathbb{R}$ mit $0 < \epsilon < \frac{1}{n}$. Zusätzlich sei nun $\alpha \in \mathbb{R}$ mit $\alpha > 4$. Wir definieren nun:

$$B'_1 := \{\alpha - 3, \alpha + 3\},$$

für $j = 2, \ldots, n$ sei

$$B'_j := \{\alpha - 1, \alpha + 1\}$$

und für $j = 1, \ldots, n$ sei

$$C'_j := \{\alpha - (1 - j\epsilon), \alpha + (1 - j\epsilon)\}.$$

Dann erhalten wir $P'_1 := B'_1 \times \ldots \times B'_n$ und $P'_2 := C'_1 \times \ldots \times C'_n$. Unsere gesamte Punktmenge P' ist nun $P' := P'_1 \cup P'_2$. Wegen $\alpha > 4$ gilt $P' \subset \mathbb{R}^n_+$.

Um unser Ziel zu erreichen werden wir die Wahl von α noch weiter einschränken. Dazu wollen wir uns noch einmal genauer mit der Bedeutung der Lemmata 4.5 bis 4.8 für den Beweis von Satz 4.4 beschäftigen. Die vier Aussagen, die dort getroffen werden, überzeugen uns davon, dass bestimmte Punktepaare aus P_2 einen Abstand kleiner als 4 zueinander haben, während alle Punktepaare aus P_1 und alle gemischten Punktepaare einen Abstand größer als 4 haben. Außerdem wird der Abstand von denjenigen Punktepaaren aus P_2 minimiert, die sich nur in der letzten Koordinate unterscheiden. Nachdem diese zu Clustern vereinigt wurden, haben dann jeweils die Clusterzentren den kleinsten Abstand, die sich nur in der letzten Koordinate unterscheiden, die ungleich 0 ist.

Aus der Konstruktion der Intervalle $B'_1, \ldots, B'_n$ und $C'_1, \ldots, C'_n$ folgt für alle Punkte $x \in P'$:

$$\alpha - 4 < x_i < \alpha + 4 \quad (i = 1, \ldots, m)$$

Für je zwei Punkte $x, y \in P'$ folgt dann aus Lemma 2.15:

$$\frac{1}{2(\alpha+4)} \parallel x - y \parallel^2 \leq D(x|y) \leq \frac{1}{2(\alpha-4)} \parallel x - y \parallel^2$$

Wir erhalten so keine genaue Darstellung für die KLD, sondern eine Eingrenzung auf ein Intervall. Aus den Lemmata 4.5 und 4.6 folgt dann mit $x, y \in P'_1$ bzw. $x \in P'_1$ und $y \in P'_2$ für die KLD nur noch

$$D(x|y) \geq \frac{1}{2(\alpha+4)} \cdot 4 = \frac{2}{\alpha+4}.$$

Aus Lemma 4.7 folgt für Punkte $x, y \in P'_1$, die sich nur in einer Koordinate j unterscheiden, für die KLD noch

$$\frac{2}{\alpha+4}(1 - j\epsilon)^2 \leq D(x|y) \leq \frac{2}{\alpha-4}(1 - j\epsilon)^2.$$

Entsprechend gilt wegen Lemma 4.8 für Punkte $x, y \in P'_1$, die sich in mindestens einer Koordinate j unterscheiden, für die KLD noch

$$D(x|y) \geq \frac{2}{\alpha+4}(1 - j\epsilon)^2.$$

Damit das Verhalten des agglomerativen Algorithmus auch mit der KLD als Abstandsmaß noch das gleiche ist wie im Abschnitt 4.3.1, dürfen sich diese Intervalle nicht überschneiden. Genau genommen müssen die beiden folgenden Ungleichungen gelten:

$$\frac{2}{\alpha-4}(1 - 1 \cdot \epsilon)^2 < \frac{2}{\alpha+4} \tag{4.5}$$

$$\frac{2}{\alpha-4}(1 - (j+1)\epsilon)^2 < \frac{2}{\alpha+4}(1 - j\epsilon)^2 \quad \text{für } j = 1, \ldots, n-1 \tag{4.6}$$

Dann können wir die verschiedenen Fälle weiterhin anhand des Abstands gegeneinander abgrenzen. Mit Ungleichung (4.5) ist auch unter der KLD der Abstand von Punktepaaren aus P_1 und gemischten Punktepaaren aus P_1 und P_2 größer als der von Punktepaaren aus P_2. Mit Ungleichung (4.6) folgt, dass die KLD von genau den Punktepaaren aus P_2 minimiert wird, die sich nur in der letzten Koordinate ungleich Null unterscheiden.

Da $4 = 4(1 - j\epsilon)$ für $j = 0$ gilt, sind (4.5) und (4.6) äquivalent zu:

$$\frac{2}{\alpha - 4}(1 - (j+1)\epsilon)^2 < \frac{2}{\alpha + 4}(1 - j\epsilon)^2 \quad \text{für } j = 0, \ldots, n-1 \tag{4.7}$$

Um dies sicher zu stellen, schränken wir die Wahl von α in Abhängigkeit von ϵ ein auf

$$\alpha > 4 + \frac{8}{2\epsilon - 2n\epsilon^2}. \tag{4.8}$$

Dass dann Ungleichung (4.7) erfüllt ist, formulieren wir im folgenden Lemma, das wir am Ende des Abschnitts beweisen.

Lemma 4.9 *Sei $\epsilon \in \mathbb{R}$, $0 < \epsilon < \frac{1}{n}$. Wählen wir ein $\alpha \in \mathbb{R}$, das die Ungleichung (4.8) erfüllt, dann ist auch Ungleichung (4.7) erfüllt.*

Wir können also zu jedem $0 < \epsilon < \frac{1}{n}$ ein $\alpha > 4$ wählen, so dass der agglomerative Clustering-Algorithmus mit der KLD als Abstandsmaß das gleiche Verhalten aufweist, das wir im Abschnitt 4.3.1 beschrieben haben.

Wenn wir nun auch die Clusteringkosten gemäß Lemma 2.15 abschätzen, erhalten wir einen im Vergleich zum Abschnitt 4.3.1 leicht veränderten Approximationsfaktor von

$$\frac{\frac{1}{2(\alpha+4)}4n - 8n^2\epsilon}{\frac{1}{2(\alpha-4)}4 + 4\epsilon + n^3\epsilon^2} = \frac{\alpha - 4}{\alpha + 4} \cdot \frac{4n - 8n^2\epsilon}{4 + 4\epsilon + n^3\epsilon^2}.$$

Wenn wir ϵ klein genug wählen, können wir den rechten Faktor wieder beliebig nahe an n heranbringen. Gleichzeitig wird α umso größer, je kleiner wir ϵ wählen. Das bedeutet, je kleiner wir ϵ wählen, desto näher bringen wir auch den linken Faktor an 1 heran. Insgesamt können wir also den Approximationsfaktor durch die Wahl eines genügend kleinen ϵ wieder beliebig nahe an n heranbringen.

Es folgt noch der angekündigte Beweis von Lemma 4.9.

Beweis (Lemma 4.9) Zunächst einmal gilt

$$\alpha > 4 + \frac{8}{2\epsilon - 2n\epsilon^2} > 4 + \frac{8(1 - (j+1)\epsilon)^2}{2\epsilon - (2j+1)\epsilon^2} \quad \text{für } j = 0, \ldots, n-1.$$

Wegen $(2j+1)\epsilon^2 - 2\epsilon < 0$ für $0 < \epsilon < \frac{1}{n}$ ist dies equivalent zu

$$8(1 - (j+1)\epsilon)^2 + (\alpha - 4)((2j+1)\epsilon^2 - 2\epsilon) < 0. \tag{4.9}$$

Nun gilt aber:

$$\begin{aligned}
& (\alpha+4)(1-(j+1)\epsilon)^2 \\
= \; & 8(1-(j+1)\epsilon)^2 + (\alpha-4)(1-(j+1)\epsilon)^2 \\
= \; & 8(1-(j+1)\epsilon)^2 + (\alpha-4)(1-2(j+1)\epsilon+(j+1)^2\epsilon^2) \\
= \; & 8(1-(j+1)\epsilon)^2 + (\alpha-4)(1-2j\epsilon-2\epsilon+j^2\epsilon^2+2j\epsilon^2+\epsilon^2) \\
= \; & 8(1-(j+1)\epsilon)^2 + (\alpha-4)(1-2j\epsilon+j^2\epsilon^2) + (\alpha-4)((2j+1)\epsilon^2-2\epsilon) \\
= \; & 8(1-(j+1)\epsilon)^2 + (\alpha-4)((2j+1)\epsilon^2-2\epsilon) + (\alpha-4)(1-j\epsilon)^2
\end{aligned}$$

Es folgt, dass Ungleichung (4.9) äquivalent ist mit

$$(\alpha+4)(1-(j+1)\epsilon)^2 \; < \; (\alpha-4)(1-j\epsilon)^2.$$

Daraus folgt unmittelbar die Ungleichung (4.7). □

4.4. Bad-Case-Szenario für den Divisive-Algorithmus

In diesem Abschnitt wollen wir Eingaben beschreiben, auf denen der Divisive-Algorithmus (siehe Abschnitt 3.2.2) kein gutes Ergebnis liefert. Für ein beliebiges $n \in \mathbb{N}$ erzeugen wir eine Multimenge im n-dimensionalen Raum $\mathbb{R}^n$, so dass das vom Divisive-Algorithmus erzeugte (2^n+1)-Clustering beliebig viel schlechter ist als die optimale Lösung. Wir untersuchen das k-Clustering-Problem mit Zentrumskosten (Problem 2.5), auf das wir uns für den Divisive-Algorithmus bereits in Kapitel 2 festgelegt haben. Als Abstandsmaß verwenden wir die L_2^2-Norm. Damit gilt

$$d(x,y) := \|x-y\|^2$$

für $x, y \in \mathbb{R}^n$. Als Clusterzentrum verwenden wir den Schwerpunkt des Clusters (siehe Definition 2.6). Die Kosten eines Clusterings sind dann also gleich der Summe der Abstände aller Punkte zu ihrem jeweiligen Clusterschwerpunkt.

Sei also $n \in \mathbb{N}$ die gewählte Dimension für unseren Punktraum. Wir beginnen unsere Konstruktion mit der Menge

$$W_0 := \{0,1\}^n.$$

W_0 enthält die 2^n Eckpunkte eines n-dimensionalen Würfels mit Kantenlänge 1. Mit W_1 bezeichnen wir eine um den Faktor $f = \frac{1}{\sqrt{2n}}$ verkleinerte Kopie von W_0, deren Punkte wir außerdem entlang jeder Koordinatenachse um den Wert $+14$

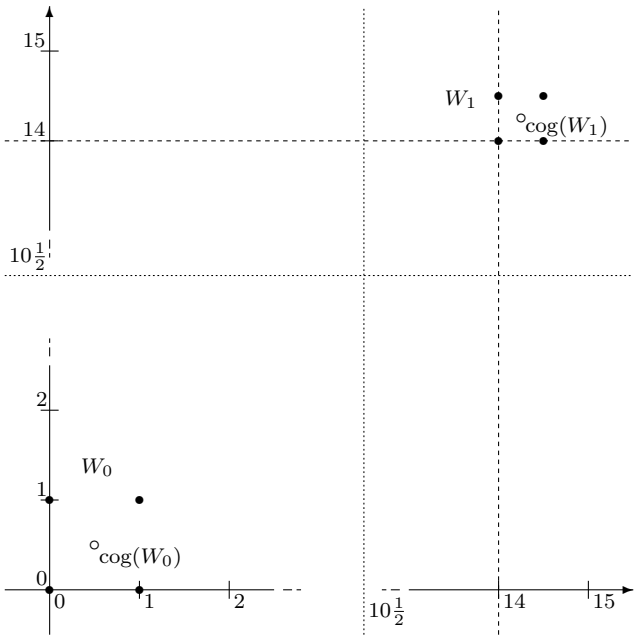

Abbildung 4.6.: Die Punktmenge $W_0 \cup W_1$ im Fall $n = 2$

verschieben. Wir definieren also

$$W_1 := \left\{ 14, 14 + \frac{1}{\sqrt{2n}} \right\}^n.$$

Abbildung 4.6 skizziert die Punktmenge $W_0 \cup W_1$ im zweidimensionalen Fall.

Als nächstes wählen wir ein $\ell \in \mathbb{N}$ mit $\ell > 2n$ und legen noch weitere $\ell - 1$ Kopien von W_1 an. Für $i = 2, \ldots, \ell$ definieren wir also

$$W_i := W_1.$$

Unsere gesamte Eingabe ist nun die Multimenge

$$P := \biguplus_{0 \le i \le \ell} W_i.$$

Man beachte, dass die Punkte aus W_1 in ℓ-facher Kopie in P enthalten sind.

Im Folgenden untersuchen wir das Verhalten des Divisive-Algorithmus auf der Menge P. Dabei bezeichnen wir mit $C_{k,i}$ den i-ten Cluster des vom Divisive-Algorithmus berechneten k-Clusterings. Welcher Cluster dabei welchen Index i erhält, hängt von der Implementierung ab und ist für unsere Untersuchungen nicht von Bedeutung.

Lemma 4.10 *Das vom Divisive-Algorithmus erzeugte* 2*-Clustering der Multimenge P besteht aus den beiden Clustern $C_{2,1} = \biguplus_{1 \le i \le \ell} W_i$ und $C_{2,2} = W_0$.*

Beweis Wir zeigen, dass ausgehend von $C_1 := C_{1,1} = P$ alle Punkte aus W_0 vom Divisive-Algorithmus in einen zweiten Cluster C_2 verschoben werden, während alle Punkte aus $P \setminus W_0$ in C_1 verbleiben.

Für das Clusterzentrum $\mathrm{cog}(C_1)$ folgt aus Definition 2.6:

$$\mathrm{cog}(C_1) \;=\; \mathrm{cog}(W_0) \;+\; \frac{\ell}{\ell+1}\big(\mathrm{cog}(W_1) - \mathrm{cog}(W_0)\big) \tag{4.10}$$

Das Clusterzentrum befindet sich also auf der Verbindungsstrecke zwischen $\mathrm{cog}(W_0)$ und $\mathrm{cog}(W_1)$ (je größer der Wert von ℓ, desto näher an $\mathrm{cog}(W_1)$). Durch die oben beschriebene Verschiebung der Punkte aus W_1 um $+14$ entlang jeder der n Koordinatenachsen und $\ell \geq 3$ (wegen $\ell > 2n$) können wir außerdem garantieren, dass gilt:

$$\mathrm{cog}(C_1) \;\in\; (10\tfrac{1}{2}, 15)^n \tag{4.11}$$

Dies folgt aus

$$\frac{3 \cdot 14 + 0}{4} \;=\; 10\tfrac{1}{2}$$

und aus

$$14 + \frac{1}{\sqrt{2n}} \;<\; 15.$$

Der Punkt $\{0\}^n$ ist daher von allen Punkten am weitesten von $\mathrm{cog}(C_1)$ entfernt. Nach Korollar 4.2 spart er am meisten Kosten ein, wenn er aus C_1 entfernt wird. Daher wird er als erstes in den neuen Cluster C_2 verschoben, wodurch sich das Clusterzentrum $\mathrm{cog}(C_1)$ ein Stück Richtung $\mathrm{cog}(W_1)$ verschiebt. Es gilt aber weiterhin $\mathrm{cog}(C_1) \in (10\frac{1}{2}, 15)^n$.

Der Divisive-Algorithmus wird nun so lange Punkte aus W_0 von C_1 nach C_2 verschieben, bis $C_2 = W_0$ gilt. Nehmen wir an, dass $C_2 \subset W_0$ gilt. Dann gilt $\mathrm{cog}(C_2) \in [0,1]^n$ für das Zentrum von C_2. Dies ist zu Beginn mit $C_2 = \big\{\{0\}^n\big\}$ gegeben.

Die Veränderung δ der Clusteringkosten beim Verschieben eines $x \in C_1$ nach C_2 können wir mit Lemma 4.3 wie folgt abschätzen:

$$\delta \;>\; d(\mathrm{cog}(C_2 \cup \{x\}), x) - 2\, d(\mathrm{cog}(C_1), x) \tag{4.12}$$

$$\delta \;<\; 2\, d(\mathrm{cog}(C_2 \cup \{x\}), x) - d(\mathrm{cog}(C_1), x) \tag{4.13}$$

Wegen $C_2 \subset W_0$ existiert ein $x_1 \in C_1 \cap W_1$. Dann ist eine einfache untere Schranke für $d(\mathrm{cog}(C_2 \cup \{x_1\}), x_1)$ gegeben durch

$$\begin{aligned} d(\mathrm{cog}(C_2 \cup \{x_1\}), x_1) \;&>\; d(\mathrm{cog}(\big\{\{1\}^n, \{14\}^n\big\}), \{14\}^n) \\ &=\; d(\{7\tfrac{1}{2}\}^n, \{14\}^n) \;>\; 42n. \end{aligned}$$

Außerdem erhalten wir eine obere Schranke für $d(\mathrm{cog}(C_1), x_1)$ durch den Abstand

zwischen $\{15\}^n$ und $\{10\frac{1}{2}\}^n$:

$$d(\mathrm{cog}(C_1), x_1) \;<\; d(\{15\}^n, \{10\tfrac{1}{2}\}^n) \;<\; 21n$$

Nach (4.12) ginge damit eine Verschiebung von x_1 nach C_1 mit zusätzlichen Kosten einher. Der Divisive-Algorithmus würde also kein $x \in C_1 \cap W_1$ in den Cluster C_2 verschieben.

Nehmen wir weiter an, dass sogar $C_2 \subsetneq W_0$ gilt, dann existiert mindestens ein $x_0 \in C_1 \cap W_0$. Da W_0 die Ecken eines n-dimensionalen Würfels mit Kantenlänge 1 beschreibt, ist eine einfache obere Schranke für $d(\mathrm{cog}(C_2 \cup \{x_0\}), x_0)$ gegeben durch

$$d(\mathrm{cog}(C_2 \cup \{x_0\}), x_0) \;<\; n.$$

Eine untere Schranke für $d(\mathrm{cog}(C_1), x_0)$ ist der Abstand zwischen $\{1\}^n$ und $\{10\frac{1}{2}\}^n$:

$$d(\mathrm{cog}(C_1), x_0) \;>\; d(\{1\}^n, \{10\tfrac{1}{2}\}^n) \;>\; 90n$$

Damit bringt ein Verschieben von x_0 nach C_1 nach (4.13) eine Kostenersparnis von mehr als $88n$.

Es wird also ein $x \in C_1 \cap W_0$ in den Cluster C_2 verschoben. Der Schwerpunkt $\mathrm{cog}(C_1)$ von C_1 entfernt sich dadurch ein Stück von x. Wiederum gilt aber $\mathrm{cog}(C_1) \in (10\frac{1}{2}, 15)^n$ wegen $x \in [0, 10\frac{1}{2}]^n$. Nach dem Verschieben von x gilt dann wieder $C_2 \subset W_0$ und solange $C_2 \neq W_0$ gilt, wiederholt sich der beschriebene Vorgang. □

Ausgangspunkt für unsere Untersuchungen ist nun das 2-Clustering des Divisive-Algorithmus, das aus den beiden Clustern $C_{2,1} = W_0$ und $C_{2,2} = \bigcup_{1 \le i \le \ell} W_i$ besteht. Wir zeigen, dass der Divisive-Algorithmus nach den nächsten $2^n - 1$ Schritten für alle Punkte aus W_0 einen eigenen Cluster erzeugt, während der Cluster $C_{2,2}$ erhalten bleibt. Wir formulieren diese Aussage im folgenden Lemma, das wir später beweisen.

Lemma 4.11 *Mit der oben definierten Punktmenge P als Eingabe erzeugt der Divisive-Algorithmus ein $(2^n + 1)$-Clustering, das 2^n Cluster mit jeweils einem der Punkte aus W_0 enthält und einen weiteren Cluster, der identisch ist mit*

$$C_{2,2} = \bigcup_{1 \le i \le \ell} W_i.$$

Die Kosten des erzeugten $(2^n + 1)$-Clustering sind dann gleich dem ℓ-fachen der Kosten von W_1. Demgegenüber hätte das $(2^n + 1)$-Clustering, das W_0 als einen Cluster erhält und jeden der 2^n Punkte aus W_1 zusammen mit seinen $\ell - 1$ Kopien in einem eigenen Cluster (mit Kosten gleich 0) zusammenfasst, die gleichen Kosten wie W_0. Da wir die L_2^2-Norm als Abstandsmaß gewählt haben und W_1 durch Stauchung um den Faktor $f = \frac{1}{\sqrt{2n}}$ aus W_0 hervorgegangen ist, gilt für die Kosten von W_0 und W_1

$$\mathrm{cost}(W_1) = f^2 \cdot \mathrm{cost}(W_0).$$

Damit ist der Approximationsfaktor der Lösung des Divisive-Algorithmus mindestens gleich

$$\ell \cdot f^2 = \frac{\ell}{2n}.$$

Da wir $\ell > 2n$ gewählt haben, besitzt die Lösung des Divisive-Algorithmus echt größere Kosten als die optimale Lösung.

Als Ergebnis dieses Abschnitts erhalten wir damit den folgenden Satz.

Satz 4.12 *Für ein beliebiges $n \in \mathbb{N}$ und ein $\ell > 2n$ existiert im n-dimensionalen Raum $\mathbb{R}^n$ eine Punktmenge der Größe $(\ell + 1)2^n$, so dass für die L_2^2-Norm als Abstandsmaß das vom Divisive-Algorithmus erzeugte $(2^n + 1)$-Clustering einen Approximationsfaktor größer $\frac{\ell}{2n}$ besitzt.* □

Es folgt abschließend noch der oben ausgelassene Beweis.

Beweis (von Lemma 4.11) Wir wollen zeigen, dass der Divisive-Algorithmus beim Berechnen des $(2^n + 1)$-Clusterings keinen Punkt aus $\bigcup_{1 \leq i \leq \ell} W_i$ in einen eigenen Cluster verschiebt.

Aufgrund der symmetrischen Würfelkonstruktion von W_1 ist die Kostenersparnis beim Verschieben in einen eigenen Cluster für alle Punkte aus W_1 (und damit auch für ihre jeweils $\ell - 1$ Kopien) gleich. Wir bezeichnen diese Ersparnis mit Γ_{W_1}. Da der Schwerpunkt von W_1 im Inneren des Würfels liegt, folgt aus Lemma 4.1: $\Gamma_{W_1} > 0$. Das Gleiche gilt für die Ersparnis beim Entfernen eines weiteren Punktes. Wir erhalten also eine echte obere Schranke für die Kostenersparnis Γ_{W_1} bei der Wahl eines Punktes aus W_1 für den neuen Cluster, wenn wir zwei beliebige Punkte aus W_1 entfernen. Wir entfernen nun also einen beliebigen Punkt aus W_1 und zusätzlich noch den Punkt aus W_1, der aus dem ersteren durch Punktspiegelung an $\mathrm{cog}(W_1)$ hervorgeht. So sorgen wir dafür, dass das Zentrum sich nicht verändert und können die Kostenersparnis ganz leicht als die Summe der Abstände der beiden entfernten Punkte zum Zentrum berechnen. Da W_1 die Eckpunkte eines Würfels mit Kantenlänge $f = \frac{1}{\sqrt{2d}}$ beschreibt, ergibt sich also

$$\Gamma_{W_1} < 2 \cdot \left\| \left\{ \frac{1}{2\sqrt{2d}} \right\}^d \right\|^2 = 2d \cdot \frac{1}{8d} = \frac{1}{4}.$$

Der Divisive-Algorithmus beginnt beim Erzeugen eines neuen Clusters immer mit dem Punkt, der am meisten Kosten einspart, wenn er aus seinem Cluster entfernt wird. Es reicht also zu zeigen, dass diese Kostenersparnis immer für einen Punkt aus W_0 größer ist als Γ_{W_1}.

Sei nun $C \subseteq W_0$ ein beliebiger Cluster von Punkten aus W_0. Nach Korollar 4.2 hat der Punkt $x \in C$, der beim Entfernen die meisten Kosten einspart, maximalen Abstand $d\big(x, \mathrm{cog}(C)\big)$ zum Clusterzentrum. Da zwei Punkte aus W_0 mindestens

den Abstand $\|1\|^2 = 1$ haben (Kantenlänge 1 des Würfels), gilt

$$d\big(x, \mathrm{cog}(C)\big) \;\geq\; \left\|\frac{1}{2}\right\|^2 \;=\; \frac{1}{4}.$$

Aus Lemma 4.1 folgt dann für die Kostenersparnis Γ_x beim Entfernen von x aus seinem Cluster:

$$\Gamma_x \;>\; \frac{1}{4} \;>\; \Gamma_{W_1}$$

□

5. Zusammenfassung

Wir haben in dieser Arbeit drei Standard-Clustering-Verfahren kennengelernt: den k-means-Algorithmus, einen einfachen agglomerativen und einen einfachen Divisive-Algorithmus. Außerdem haben wir uns mit mehreren Abstandsmaßen beschäftigt. Von besonderem Interesse war dabei die Kullback-Leibler-Divergenz, die ein Abstandsmaß für Wahrscheinlichkeitsverteilungen darstellt.

Mit Hilfe der C++-Implementierung, die im Rahmen dieser Arbeit entstanden ist, haben wir die vorgestellten Algorithmen miteinander verglichen. Dabei stellte sich überraschenderweise heraus, dass der einfachste und schnellste Algorithmus auch die besten Ergebnisse liefert, bezogen auf die Kosten des berechneten Clusterings. Diese Beobachtung wäre es sicherlich wert, eingehender untersucht zu werden.

Im Fokus der Analyse stand aber die Konstruktion von Bad-Case-Szenarien für den agglomerativen und den Divisive-Algorithmus. Für den untersuchten agglomerativen Algorithmus konnte mit der Verallgemeinerung der Kullback-Leibler-Divergenz als Abstandsmaß auf dem $\mathbb{R}^n_+$ eine untere Schranke für den Approximationsfaktor gezeigt werden. Diese Schranke lässt sich für eine Punktmenge der Größe 2^{n+1} beliebig nahe an n heranbringen. Für den Divisive-Algorithmus konnte ein ähnliches Ergebnis erreicht werden. Hier wurde für die quadrierte L_2-Norm als Abstandsmaß auf dem $\mathbb{R}^n$ eine untere Schranke für den Approximationsfaktor bewiesen. Dazu wurde für ein $\ell \in \mathbb{N}$ mit $\ell > 2n$ eine Punktmenge der Größe $(\ell+1)2^n$ konstruiert, aus der wir $\frac{\ell}{2n}$ als untere Schranke für den Approximationsfaktor erhalten.

Da sich die verallgemeinerte Kullback-Leibler-Divergenz nach unten und nach oben durch die quadrierte L_2-Norm abschätzen lässt, ist das zweite Ergebnis durchaus interessant. Man darf optimistisch sein, dass sich die bewiesene untere Schranke auch auf die verallgemeinerte Kullback-Leibler-Divergenz übertragen lässt.

Für eine weitere Erforschung bietet sich aber vor allem die echte Kullback-Leibler-Divergenz für Wahrscheinlichkeitsverteilungen an. Dort sind wegen des aufgezeigten Bezuges zur Codierungstheorie ähnliche Ergebnisse wünschenswert, wie die für die verallgemeinerte Variante bewiesenen Schranken.

A. Anhang

A.1. Implementierung

Im Rahmen dieser Diplomarbeit ist auch eine C++-Implementierung der in Kapitel 3 vorgestellten Clustering-Algorithmen entstanden. Die Sammlung von Klassen trägt den Namen *CluE*, der für CLUstering Environment steht. Zusätzlich zu den Basisklassen mit der eigentlichen Funktionalität gehört dazu auch ein Testprogramm mit grafischer Benutzeroberfläche. Es diente dazu, die verschiedenen Algorithmen miteinander zu vergleichen und half auch bei der Erarbeitungen der Bad-Case-Szenarien aus Kapitel 4.

Durch den Einsatz von Templates und den objektorientierten Entwurf ist es gelungen, die Implementierung der Algorithmen komplett von den Abstands- und Kostenfunktionen, den Verfahren zur Berechnung der Clusterzentren und den zu clusternden Datentypen zu trennen. Durch die Verwendung eines Templateparameters können die Clustering-Algorithmen theoretisch an jeden beliebigen Datentyp angepasst werden. Falls ein Algorithmus eine Abstands- oder eine Kostenfunktion benötigt oder auf die Berechnung von Clusterzentren angewiesen ist, müssen nur entsprechende Klassen implementiert werden, die die geforderte Funktionalität für den gewünschten Datentyp bereitstellen. So erwartet beispielsweise die Implementierung des agglomerativen Algorithmus unter anderem eine Abstandsfunktion als Parameter im Konstruktor.

In der CluE-Implementierung repräsentieren die Instanzen einer Algorithmenklasse bereits das Ergebnis. Der Clustering-Algorithmus selbst wird im Konstruktor umgesetzt. Man übergibt dem Algorithmus im Konstruktor die zu clusternden Objekte, evtl. benötigte Abstands- und Kostenfunktionen oder andere Hilfsobjekte, z. B. zur Berechnung der Clusterzentren. Über die fertige Instanz gelangt man dann an das berechnete Clustering.

Wir stellen zunächst die Basisklassen vor.

A.1.1. Basisklassen

`Clustering<T>` Diese Template-Klasse repräsentiert einen Clustering-Algorithmus. Methoden zum Auslesen des Ergebnisses sind hier bereits vorgegeben und werden von spezialisierenden Algorithmenklassen implementiert. Der Template-Parameter `T` steht für den zu clusternden Datentyp und wird von einer Algorithmenklasse normalerweise nicht überschrieben. Dadurch lässt sich der Algorithmus für jeden beliebigen Datentyp einsetzen.

`Measure<T>` Diese Template-Klasse kapselt eine Abstands- und eine Kostenfunktion. Der Template-Parameter `T` steht wieder für den zu clusternden Datentyp. Spezialisierende Klassen implementieren eine Abstands- und eine Kostenfunktion für einen speziellen Datentyp, d. h. hier soll der Template-Parameter überschrieben werden. Dadurch können die Clustering-Algorithmen, wenn sie mit einem `Measure`-Objekt arbeiten, durch eine ensprechende Implementierung an jeden Datentyp angepasst werden.

`Substitution<T>` Diese Template-Klasse kapselt die Funktionalität der Zentrumsberechnung. Die enthaltene Methode erwartet einen Vektor von Objekten der Template-Klasse und gibt auch ein Objekt der Template-Klasse zurück. Wie bei der `Measure`-Klasse können so die Clustering-Algorithmen mit einer entsprechenden `Substitution`-Implementierung an jeden Datentyp angepasst werden.

A.1.2. Bereits unterstützte Datentypen

Für das oben beschriebene Testprogramm wurden bereits einige Datentypen mit dazugehörigen `Measure`- und `Substitution`-Klassen implementiert. Die beiden wichtigsten sollen hier kurz vorgestellt werden. Abbildung A.1 zeigt ein Klassendiagramm mit allen erwähnten Klassen.

`Point` Die Klasse `Point` speichert einen beliebig-dimensionalen Punkt mit `double`-Koordinaten und einem `double`-Gewicht. Mit der Klasse `CenterOfGravity` wurde eine Spezialisierung der oben beschriebenen `Subtitution<T>`-Klasse geschaffen, die den Template-Parameter `T` durch `Point` ersetzt. `CenterOfGravity` kann den gewichteten Schwerpunkt einer Menge von Punkten berechnen. Außerdem wurden in gleicher Weise mehrere Abstands- und Kostenfunktionen implementiert, die die Klasse `Measure<T>` spezialisieren. Alle implementieren als Abstandsfunktion ein Abstandsmaß, das auch der Kostenfunktion zugrunde gelegt wird. Als Kostenfunktion wird die Funktion cost_{ZK} aus Problemdefinition 2.5 implementiert. Es handelt sich um die folgenden fünf Klassen für die angegebenen Abstandsmaße:

- `L1norm`: implementiert die L_1-Norm als `Measure<Point>`
- `L1sq`: implementiert die quadrierte L_1-Norm als `Measure<Point>`
- `L2norm`: implementiert die L_2-Norm als `Measure<Point>`
- `L2sq`: implementiert die quadrierte L_2-Norm als `Measure<Point>`
- `PtKullbackLeibler`: implementiert die verallgemeinerte Kullback-Leibler-Divergenz als `Measure<Point>`

`ProbabilityDistribution` Die Klasse `ProbabilityDistribution` speichert eine beliebig-dimensionale Wahrscheinlichkeitsverteilung mit `double`-Wahrscheinlichkeitswerten und einem `double`-Gewicht. Auch hierzu gibt es eine entsprechende `Subtitution<ProbabilityDistribution>`-Implementierung mit

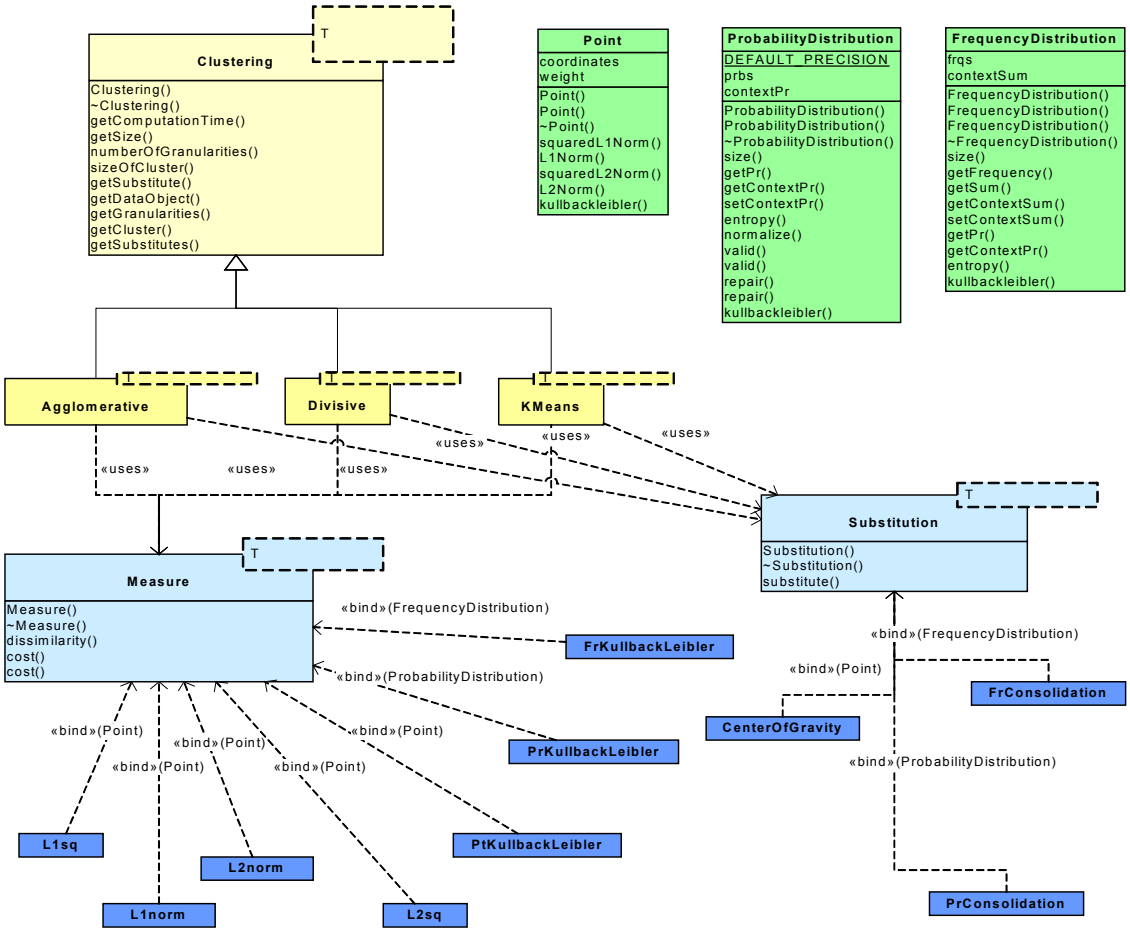

Abbildung A.1.: Klassendiagramm für die CluE-Implementierung

dem Namen `PrConsolidation`. Diese berechnet das gewichtete Mittel von Wahrscheinlichkeitsverteilungen. Für die Abstands- und Kostenfunktionen wurde eine `Measure<ProbabilityDistribution>`-Klasse mit Namen `PrKullbackLeibler` implementiert. Sie berechnet die Kullback-Leibler-Divergenzen, wie in Abschnitt 4.2 beschrieben.

Analog zur Klasse `ProbabilityDistribution` wurde auch noch die Klasse `FrequencyDistribution` für Häufigkeitsverteilungen implementiert. Sie bietet im Prinzip die gleiche Funktionalität, arbeitet aber mit Häufigkeiten und kommt daher ohne `double`-Werte aus. Passend dazu gibt es die entsprechenden Klassen `FrConsolidation` und `FrKullbackLeibler`.

A.1.3. Implementierte Algorithmen

Alle drei in Kapitel 3 beschriebenen Algorithmen sind in CluE enthalten. Jeder ist in Form einer `Clustering<T>`-Spezialisierung implementiert. Es handelt sich dabei um die Klassen `Agglomerative<T>`, `Divisive<T>` und `KMeans<T>`. Im Konstruktor erwarten alle ein `Measure<T>`- und ein `Substitution<T>`-Objekt um ihre Berechnungen durchführen zu können.

Literaturverzeichnis

[Ackermann u. a. 2007] Ackermann, M. R. ; Blömer, J. ; Sohler, C.: *Clustering for Metric and Non-Metric Distance Measures*. Juli 2007. – unpublished

[Arthur und Vassilvitskii 2006] Arthur, D. ; Vassilvitskii, S.: How Slow is the k-Means Method? In: Amenta, Nina (Hrsg.) ; Cheong, Otfried (Hrsg.): *Proceedings of the 22nd ACM Symposium on Computational Geometry, Sedona, Arizona, USA, June 5-7, 2006*, ACM, 2006, S. 144–153

[Dasgupta und Long 2005] Dasgupta, S. ; Long, P. M.: Performance Guarantees for Hierarchical Clustering. In: *JCSS: Journal of Computer and System Sciences* 70 (2005), Nr. 4, S. 555–569

[Day und Edelsbrunner 1984] Day, W. H. E. ; Edelsbrunner, H.: Efficient Algorithms for Agglomerative Hierarchical Clustering Methods. In: *Journal of Classification* 1 (1984), Dezember, Nr. 1, S. 7–24

[Foucault 2002] Foucault, M.: *Die Ordnung der Dinge*. Suhrkamp, 2002

[Guénoche u. a. 1991] Guénoche, A. ; Hansen, P. ; Jaumard, B.: Efficient Algorithms for Divisive Hierarchical Clustering with the Diameter Criterion. In: *Journal of Classification* 8 (1991), Januar, Nr. 1, S. 5–30

[Hoppe 2003] Hoppe, U.: *Markowprozesse zur Modellierung bei Codekompression*, Universität Paderborn, Institut für Informatik, Diplomarbeit, November 2003

[Kullback 1959] Kullback, S.: *Information Theory and Statistics*. Wiley, 1959

[Kullback und Leibler 1951] Kullback, S. ; Leibler, R. A.: On Information and Sufficiency. In: *The Annals of Mathematical Statistics* 22 (1951), März, Nr. 1, S. 79–86

[MacQueen 1967] MacQueen, J.: Some Methods for Classification and Analysis of Multivariate Observations. In: Le Cam, L. M. (Hrsg.) ; Neyman, J. (Hrsg.): *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability* Bd. 1. Berkeley, Califonia : University of California Press, 1967, S. 281–297

[Mahalanobis 1936] Mahalanobis, P. C.: On Generalized Distance in Statistics. In: *Proceedings of the National Inst. Sci. (India)* 12 (1936), S. 49–55

[Mercer 2003] Mercer, D. P.: Clustering Large Datasets / Linacre College. Oktober 2003. – Forschungsbericht.

[Sayood 2005] Sayood, K.: *Introduction to Data Compression*. Third. Elsevier, 2005. – 704 S

[Shannon 1948] Shannon, C. E.: A Mathematical Theory of Communication. In: *The Bell System Technical Journal* 27 (1948), Nr. 3, S. 379–423